Internationaler Studiengang Medieninformatik B.Sc.
Fakultät für Elektrotechnik und Informatik
Neustadtswall 30
28199 Bremen

Web GIS Erweiterung des CMS Joomla! im Kontext der Biodiversität

Eine Web GIS Erweiterung für das Open Source Content Management Systems Joomla! zur Erfassung und Visualisierung von Biodiversitätsdaten mit Bezug auf „User Generated Content“.

Bachelor-Arbeit
von
Carl-Heinz Genzel

vorgelegt bei
Prof. Dr.- Ing. Heide-Rose Vatterrott
und
Prof. Dr. Heiko Brunken

Bremen, Februar 2011

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Abstract

Within a project of the International Degree Course Industrial and Environmental Biology at the University of Applied Sciences Bremen, a Joomla!-based information system to list fish species and their occurrences with a focus on biodiversity and water pollution control, was developed between 2006 and 2008.

Since then, *Geographic Information Systems* (*GIS*) have been further established in the Web. To utilize improvements related to *GIS* and because of new application fields, a reimplementation has become necessary. That's why, the main aim of this bachelor thesis is the development of a Joomla!-based web application which in general can be used for different biodiversity groups and their occurrences. Therefor old functionalities need to be preserved whereas improvements have to be made by adding new functionalities. Thereby it should be possible to add the resulting application to a basic Joomla! system. Once installed, it has to be possible to list and display different species and their occurrences using the application. Because the application basically has the form of an atlas, the focus lies on the visualisation and collection of spatial data related to species. In the future the application should be useful to cover different biodiversity groups.

Keywords: Joomla!, *Geographic Information System*, *GIS*, Biodiversity, Atlas, Spatial Data, Web Application

Zusammenfassung

In einem Projekt des Internationalen Studiengangs „Technische und Angewandte Biologie“ an der Hochschule Bremen, entstand zwischen 2006 und 2008 ein Joomla!-basiertes Fachinformationssystem verbunden mit Geoinformationen zur Feststellung von Fischarten und deren Vorkommen mit dem Schwerpunkt *Biodiversität* und Gewässerschutz.

Seitdem hat sich das Thema *Geographic Information System* (*GIS*) weiter im Web etabliert. Aufgrund neuer Möglichkeiten im *GIS*-Bereich und neuen Einsatzgebieten ist die Entwicklung einer Joomla!-basierten Webanwendung, die generisch für verschiedene Lokationen und Biodiversitätsgruppen eingesetzt werden kann, Ziel dieser Arbeit. Dabei sollen die aus den vorhergehenden Projekten zu diesem Thema entwickelten Anwendungsteile zu einer Joomla!-Anwendung zusammenfügt und um fehlende Kernfunktionen erweitert werden, sodass die Integration dieser Anwendung in eine Joomla!-Grundinstallation mit geringem Aufwand möglich ist. Aufgabe der Anwendung ist die Erfassung und Darstellung von unterschiedlichen Biodiversitätsdaten und deren Vorkommen. Dabei geht es um eine Umsetzung nach Art eines Atlas, weshalb der Schwerpunkt der Anwendung auf der Visualisierung und Erfassung von Geodaten mit Bezug auf *Biodiversität* liegt. Die Anwendung soll zukünftig für verschieden Biodiversitätsgruppen einsetzbar sein.

Schlüsselbegriffe: Joomla!, *Geographic Information System*, *GIS*, *Biodiversität*, Atlas, Räumliche Daten, Webanwendung

Inhalt

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Aufgabe/Zielsetzung . . . 2
- 1.3 Gliederung . . . 3

2 Anforderungsanalyse **5**
- 2.1 Funktionale Anforderungen . . . 6
  - 2.1.1 Allgemeine Informationen . . . 6
  - 2.1.2 Geographische Informationen . . . 9
- 2.2 Technische Anforderungen . . . 12
  - 2.2.1 Allgemeine Informationen . . . 12
  - 2.2.2 Geographische Informationen . . . 13
  - 2.2.3 Software Vorgaben . . . 13
- 2.3 Begrenzungen . . . 13

3 Aktuelle Methoden und Technologien **15**
- 3.1 Webdarstellung . . . 15
  - 3.1.1 Joomla . . . 16
  - 3.1.2 OpenLayers und GeoExt . . . 20
- 3.2 Datenverwaltung . . . 22
  - 3.2.1 MySQL . . . 22
  - 3.2.2 PostgreSQL . . . 25
- 3.3 Geoinformationen . . . 27
  - 3.3.1 PROJ.4 . . . 27
  - 3.3.2 Kartenquellen . . . 28
- 3.4 Folgerungen . . . 33
- 3.5 Lösungsansätze . . . 36

4 Realisierungskonzept **39**
- 4.1 Frontend . . . 40
- 4.2 Backend . . . 42
- 4.3 Plugin . . . 44

5 Realisierung **45**
- 5.1 Datenbank . . . 46
  - 5.1.1 Species . . . 48
  - 5.1.2 Location . . . 49

5.2 Webanwendung 49
5.2.1 Nutzermodell 50
5.2.2 Backend 51
5.2.3 Frontend 67
5.2.4 Plugin 81

6 Test und Evaluation **85**
6.1 Tests 85
6.1.1 Entwicklertest 85
6.1.2 Nutzertest 85
6.2 Evaluation 86
6.2.1 Entwicklertest 86
6.2.2 Nutzertest 86

7 Bewertung und Ausblick **87**

Glossar **91**

Abbildungsverzeichnis **99**

Tabellenverzeichnis **101**

Literaturverzeichnis **103**

Anhang

A Digitales Medium **107**

1 Einleitung

1.1 Motivation

Seit dem Start von Google Maps im Februar 2005[HSB19] hat sich das Thema *Geographic Information System* (*GIS*) im Web etabliert. Was anfangs noch skeptisch betrachtet wurde, findet heute in den verschiedensten Bereichen Anwendung. [TAWAE08] Ob zur Abfrage von Entfernungen zwischen Orten, als Spiel mit Koordinaten[1] oder zur Lagebeschreibung. Webbasierte Geoinformationssysteme, kurz Web*GIS*, sind in kleinen Blogs bis hin zu wissenschaftlichen Darstellungen auf einschlägigen Webseiten zu finden. Dies liegt nicht zuletzt daran, dass der Umgang mit diesen Systemen und den nötigen Daten, dank den Standardisierungen durch das *Open Geospatial Consortium*[2], vereinfacht worden ist.

In einem Projekt des Internationalen Studiengangs „Technische und Angewandte Biologie“[3] an der Hochschule Bremen unter Leitung von Prof. Dr. Heiko Brunken entstand zwischen 2006 und 2008 eine erste Umsetzung eines Fachinformationssystems zur Feststellung von Fischarten und deren Vorkommen mit dem Schwerpunkt *Biodiversität* und Gewässerschutz. In dem Projekt wurde mit Hilfe von *GIS* eine Möglichkeit geschaffen Biodiversitätsdaten, in diesem Fall Fischarten, aus verschiedenen Quellen zu erfassen und geographisch zu visualisieren, um diese Informationen Fischkundlern und weiteren Interessierten zur Verfügung stellen zu können.
Dieser digitale Fischartenatlas ist frei über das Internet zugänglich und basiert auf dem Open Source *Content Management System* Joomla!.[HSB19]

In der ersten Umsetzung wurden hauptsächlich Daten verschiedener Institute und Ämter aus Deutschland und Österreich erfasst. Daraus resultierte eine bis dato als vollständig zu betrachtende Erfassung der Fischarten.[HSB19] Diese Vollständigkeit muss jedoch nicht für die quantitative Erfassung der Vorkommen gelten. Im Rahmen der mit „Web 2.0“ verknüpften Entwicklungen im Bereich „*User Generated Content*“ wäre es denkbar, neben den offiziellen Stellen, durch privat interessierte Personen zusätzliche Daten sammeln zu können.

[1] http://www.geocaching.de/index.php?id=74
[2] http://www.opengeospatial.org/
[3] http://www.hs-bremen.de/internet/de/studium/stg/istab/index.html

Die derzeitige Implementierung verlangt jedoch einen erhöhten Aufwand, diese Daten in das System zu importieren. Es besteht zum jetzigen Zeitpunkt nur die Möglichkeit zu erfassende Daten an ein Team von Fachkräften zu liefern, die dieses Projekt unterstützen. Die Fachkräfte fügen diese Daten, nach einer Überprüfung, manuell mit Hilfe von MySQL in das *CMS* ein.[HSB19] Dies ist neben der Fehleranfälligkeit bei direkten Eingaben in eine Datenbank, sehr zeitaufwändig und erfordert Fachkenntnisse im Umgang mit MySQL. Es erschwert besonders den Zugang für außenstehende Personen, da immer eine direkte Kontaktaufnahme nötig ist.

Neben der Erfassung von Daten sind auch in der Darstellung der Daten Defizite zu erkennen, die die Nutzbarkeit des Angebots einschränken. Derzeit wird eine statische Karte mit Markierung an den Orten, an denen Fischarten festgestellt wurden, dargestellt. Dies ist, aufgrund der angesprochenen Weiterentwicklungen im *GIS*-Bereich, nicht mehr zeitgemäß und lässt auch keine genauere Ansicht oder Auswertung der Daten zu. Die Nutzbarkeit der Daten wird dadurch erheblich beeinträchtigt. Unabhängig von den Markierungen der Fischvorkommen sind die angezeigten Karten auch nicht zwingend aktuell und zusätzlich nicht skalierbar, da sie als Bilddaten auf einem Server liegen.[HSB19] Serviceangebote wie *Open Street Map* (*OSM*)[4] oder Google Maps[5] sind meist aktueller und in verschiedenen Auflösungen nutzbar.

1.2 Aufgabe/Zielsetzung

Wie im Abschnitt „Motivation“ beschrieben, gibt es einige Stellen, an denen die erste Version Schwächen aufweist. Hinzu kommt, dass es derzeit für die verschiedenen Einsatzorte verschiedene Datenmodelle gibt, strukturgleiche Daten werden also in verschiedenen Formen und Datenbanken gespeichert. Das führt dazu, dass die Webanwendung auf verschiedene Datenbanken zurückgreifen muss und somit keine einheitliche Verarbeitung möglich ist.

Aus diesem Grund lautet die Aufgabe dieser Arbeit eine Joomla!-basierte Webanwendung zu entwickeln, die generisch für verschiedene Lokationen und Biodiversitätsgruppen eingesetzt werden kann. Mit dieser Anwendung soll Nutzern die Möglichkeit gegeben werden, Vorkommen von *Biodiversitäten* zu erfassen und abzufragen. Da es um eine Umsetzung nach Art eines Atlas geht, steht vor allem die Visualisierung und Erfassung von Geodaten im Vordergrund. Ein zweiter wichtiger Punkt ist der Umgang mit den Informationen. Es muss ein Nutzermodell gestaltet werden, das spezielle Rechte für existierende Nutzertypen und deren Fachkenntnisse sowie deren Vertrauenswürdigkeit berücksichtigt.

[4] http://www.openstreetmap.de/faq.html#wie_daten_nutzen
[5] http://code.google.com/intl/de-DE/apis/maps/documentation/javascript/

1.3 Gliederung

Diese Arbeit gliedert sich, abgesehen von der Einleitung, in sechs unterschiedliche Teile. Im ersten Teil wird die Ausgangslage, mit Blick auf die zu erreichenden Ziele, beschrieben. Hierzu wird eine Analyse der Anforderungen vorgenommen.

Im zweiten Teil dieser Arbeit werden die nötigen Techniken und Methoden untersucht, die zum Erfolg dieser Arbeit beitragen könnten. Dieser Teil gliedert sich in die Abschnitte Webdarstellung, Datenhaltung und Geoinformationen. Er gibt darüber Aufschluss, welche Möglichkeiten für eine Umsetzung dieser Arbeit vorhanden sind. Als Resultat werden nach einer kurzen Folgerung einzelne Lösungsansätze vorgestellt und untersucht.

In Teil Drei der Arbeit werden, im Rahmen eines Konzeptes, erste Ansätze für die tatsächliche Realisierung der Anwendung festgehalten. Dazu werden einzelne Anforderungen für die Implementierung strukturiert und in Beziehung zu den untersuchten Techniken und Methoden gesetzt.

Der vierte Teil der Arbeit befasst sich mit der Realisierung der Webanwendung. Hier wird auf konkrete Aspekte der Realisierung eingegangen. Die Beschreibung der Webanwendung gliedert sich in die Joomla!-typischen Bereiche Backend und Frontend. Diese Beschreibung verdeutlicht die grundlegende Struktur und Arbeitsweise der Webanwendung sowie die damit zusammenhängende Vorgehensweise bei der Implementierung.

Während der Arbeit wurden kleinere Tests durchgeführt, der fünfte Teil erläutert diese und fasst die entstandenen Resultate zusammen.

Im letzten Abschnitt werden die umgesetzten Funktionen noch einmal zusammengefasst, um letztendlich ein Urteil und einen Ausblick geben zu können.

Im Verlauf der Arbeit wird es immer wieder vorkommen, dass Begriffe genutzt werden, die im Rahmen dieser Arbeit nicht weiter erläutert werden können. Nötige Erklärungen oder hilfreiche Hinweise zu diesen Begriffen sind in einem Glossar am Ende der Arbeit erfasst. Begriffe die im Glossar erfasst sind, werden in leicht geneigter Schrift dargestellt. In dieser Arbeit wird außerdem, auf im Internet vertretene Ressourcen aufmerksam gemacht. Zur Verbesserung der Lesbarkeit sind diese Ressourcen nur namentlich genannt und mit einer Fußnote markiert. Am Ende einer jeden Seite sind die, den jeweiligen Fußnoten zugehörigen Internetadressen, angegeben.

2 Anforderungsanalyse

Zu Beginn dieser Arbeit wurde im Rahmen einer Beratung des Projektteams, bestehend aus Prof. Dr. Heide-Rose Vatterrott, Prof. Dr. Heiko Brunken, Martin Winkler, Larissa Fittjer und Konstanze Steinhausen, über die zukünftige Entwicklung des Biodiversitätenprojektes gesprochen. Das Ergebnisprotokoll dieser Beratung ist auf der, im Anhang dieser Arbeit, beigelegten CD zu finden. Es bildet die Grundlage für die beschriebenen Vorgaben, Probleme und Strukturen.

Aus der Zielsetzung dieser Arbeit geht hervor, dass ein digitaler Atlas zur Erfassung und Visualisierung von Biodiversitätsdaten mit einem möglichst generischen Ansatz implementiert werden soll. In diesem Atlas soll es berechtigten Nutzern möglich sein, Fundorte biologischer Arten einzutragen, diese Fundorte werden auf einer Karte visualisiert. Zusätzlich soll es die Möglichkeit geben Informationen und Bilder einer Art abzurufen. Die Implementierung soll so erfolgen, dass die Möglichkeit besteht, ein Joomla!-Grundsystem um die angesprochene Atlasfunktionalität zu erweitern.

Daraus resultieren zwei Aufgabenbereiche. Zum einen geht es um die Verwaltung von Informationen im Allgemeinen, hierzu gehört unter Anderem die Erfassung von Biodiversitätsdaten. Zum Anderen geht es um die Verwendung von Methoden aus dem Bereich *GIS*, zur Darstellung und Auswertung von geographischen Daten im Bezug auf die Biodiversätsdaten.

Im Rahmen der Beratung sind neben funktionalen auch technische Anforderungen entstanden. Diese Anforderungen werden nicht durch das erwähnte Nutzermodell beeinflusst, da sie nur im Bezug auf die Implementierung interessant sind. Aus diesem Grund wird die Anforderungsanalyse in funktionale Anforderungen und technische Anforderungen gegliedert. In beiden Bereichen wird zusätzlich der Bezug auf allgemeine Informationen und geographische Informationen unterschieden.

Um möglichst wenig an den Arbeitsgewohnheiten der bisherigen Nutzer zu ändern, wird versucht, alte Funktionen, sofern möglich und nötig, in die neue Komponente zu integrieren. Übernommene Funktionen werden diesbezüglich gekennzeichnet.

2.1 Funktionale Anforderungen

Die im Folgenden erläuterten Anforderungen haben unterschiedliche Auswirkungen auf die, der Anwendung zugrunde liegenden Daten. Um die Integrität der Daten garantieren zu können, ist ein Nutzermodell erforderlich, dass Nutzer aufgrund ihrer Autorität wie in Abbildung 2.1 zu sehen in verschiedene Kategorien des Vertrauens unterteilt.

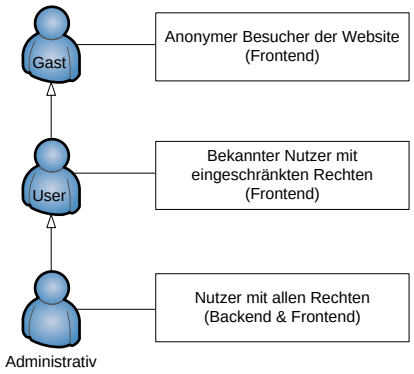

Abbildung 2.1: Nutzermodell auf Basis des Vertrauens

Zur besseren Übersicht werden die einzelnen Anforderungen daher, soweit dies möglich ist, den einzelnen Nutzertypen zugeordnet. Prinzipiell gilt jedoch, dass administrativen Nutzern der Zugang zu allen Daten erlaubt ist.

2.1.1 Allgemeine Informationen

Durch die Verbindung von geographischen Daten mit Biodiversitätsdaten besteht eine Hauptaufgabe der Anwendung in der Erfassung von allgemeinen Informationen zur Darstellung. Daraus resultieren zwei Schwerpunkte, es sollen Biodiversitätsdaten erfasst und dargestellt werden können, es sollen aber auch ergänzende Informationen zu den geografischen Daten der Biodiversitätsvorkommen erfasst und dargestellt werden. Da die Erfassung von Biodiversitätsdaten, also den Artinformationen, Fachkenntnisse voraussetzt, ist diese Funktion nur für administrative Nutzer erreichbar. Dies war schon in der vorhergehenden Umsetzung der Fall.

In der ursprünglichen Umsetzung konnte eine Art mit einem Fachnamen und Populärnamen erfasst werden. Es konnten ein Bearbeiter und die Person, die diese Art erstmalig bestimmt hat, inklusive eines Bestimmungsjahres eingetragen werden. Ein taxonomischer Rang, ein Monografiedokument, sowie ein Vorschaubild konnten jeder Art beigefügt werden. Ein Freitext ist, neben der Festlegung ob die Art ein Synonym einer anderen Art ist oder einer Revision unterliegt, ebenfalls möglich. Ein weiterer Bereich hilft bei der Erfassung der Bearbeiter, die der Bearbeitung einzelner Arten zugeteilt werden können. Jeder Bearbeiter kann mit

Namen, den ihm zugeteilten Arten, einer E-Mail-Adresse und einer Webseite erfasst werden. Die ursprüngliche Umsetzung hat es den administrativen Nutzern auch ermöglicht, jede Art mit einem Gefährdungsstatus den wichtigsten roten Listen zuzuordnen.

Diese angesprochenen Funktionen wurden in die Anforderungen für die neue Komponente übernommen. Einzig die Verlinkung auf eine Artmonographie wurde nicht übernommen. Um die Bearbeitung der Artmonographie zu vereinfachen, wird stattdessen eine Möglichkeit angedacht, mit der Anwendung entsprechenden Freitext erfassen zu können.

Zu der Erfassung einer Art soll es durch die neue Anwendung möglich sein, eine Art mit verschiedenen, mehrsprachigen Populärnamen zu versehen. Um außerhalb wissenschaftlicher Einordnungen weitere Gruppierungen innerhalb der Biodiversitäten vornehmen zu können, soll es möglich sein, Arten mit Metadaten zu verbinden. Dazu wären unbestimmte Angaben wie zum Beispiel Salz- oder Süßwasserfisch im Rahmen des Fischatlas denkbar.

Eine weitere Ergänzung stellt die Erfassung der *Taxonomie* dar. Es ist gefordert, eine Art innerhalb der biologischen *Taxonomie* einordnen zu können. Da sich die biologische *Taxonomie* jedoch aufgrund von neuen Erkenntnissen verändert, kann diese nicht fest in der Anwendung verankert werden. Aus diesem Grund sind Funktionen nötig, die es ermöglichen die *Taxonomie* zu verwalten und neue Einträge vornehmen zu können, gleichfalls muss es natürlich auch möglich sein, Einträge zu löschen oder zu verschieben.

Zusätzlich zur Erfassung von Arten und den dazugehörigen Daten soll es administrativen Nutzern möglich sein, diese Daten auch zu verwalten. Zur Verwaltung gehört das Löschen und Ändern der Datensätze.

Den zweiten Teil der allgemeinen Informationen stellen die ergänzenden Informationen zu einem Vorkommen einer Art dar. Da dieser Teil nicht in Form einer Anwendung realisiert wurde, wurden auch keine Funktionen aus der alten Umsetzung übernommen.

Was die Darstellung von Arten betrifft, soll dieser Teil der Anwendung für alle Nutzer zugänglich sein. Also auch Nutzer mit dem Status Gast haben Zugriff auf die Informationen. Um eine Art anzuzeigen, muss sie wie auch in der ursprünglichen Umsetzung von einem Nutzer über eine Liste ausgewählt werden. Diese Liste soll in mindestens zwei Versionen anzeigbar sein. Zum einen soll eine Liste mit alphabetisch sortierten lateinischen Fachnamen und zum anderen eine Liste mit sortierten Populärnamen angezeigt werden. Sollten Populärnamen in mehreren Sprachen verfügbar sein, soll für jede vorhandene Sprache eine Liste von Populärnamen dargestellt werden. In Verbindung mit einer Karte, auf die im

Bereich „Geographische Informationen“ genauer eingegangen wird, sollen nach der Auswahl einer Art alle eingetragenen Informationen dieser Art dargestellt werden.

Im Gegensatz zu den Artinformationen soll die Erfassung von Orten im Rahmen des „*User Generated Content*“ auch anderen Personen möglich gemacht werden. Dennoch ist sie nicht für jeden zugänglich, da die Erfassung von Orten für Vorkommen, wie die Erfassung von Arten, einem höheren Qualitätsanspruch unterliegt. Aus diesem Grund soll die Erfassung von Vorkommen nur für registrierte und administrative Nutzer möglich sein.

Jeder Nutzer dieser Gruppen soll in der Lage sein, ein Vorkommen mit einem Namen und einem Funddatum erfassen zu können. Zusätzlich wird ein Nutzer gezwungen das Vorkommen, zu einer Art zuzuordnen. Dazu soll er eine Art anhand ihres lateinischen Namens auswählen können. Jede Eintragung soll automatisch mit dem Nutzer verknüpft werden, der diese Eintragung vorgenommen hat. Ein administrativer Nutzer soll die Möglichkeit bekommen, Vorkommen im Namen eines anderen Nutzers eintragen zu können, er soll in der Lage sein, einen beliebigen Nutzer mit dem von ihm eingetragenen Vorkommen zu verknüpfen. Jedem Nutzer soll weiterhin die Möglichkeit gegeben werden ein Kommentar zu hinterlassen. Er soll diesbezüglich selbst entscheiden können, ob er diesen Kommentar veröffentlicht oder nur zur Einsicht für administrative Nutzer verfügbar macht. Bevor der Nutzer einen Eintrag endgültig einfügt, soll es ihm möglich sein, rechtliche Einstellungen zu tätigen.

Während neu eingetragenen Vorkommen von administrativen Nutzern vollständig vertraut wird, sollen die Eintragungen von registrierten Nutzern erst nach der Freigabe durch einen administrativen Nutzer in der Anwendung verwendet werden dürfen. Um die Freigabe zu vereinfachen, ist eine Benachrichtigungsfunktion gewünscht, die eine Nachricht versendet, nachdem ein neuer Datensatz von einem registrierten Nutzer eingefügt wurde. Unnötige Kontrollen auf neue Datensätze sollen dadurch vermieden werden. Zusätzlich zur Erfassung von Vorkommen, soll es administrativen Nutzer möglich sein, eingetragene Vorkommen zu verwalten. Zur Verwaltung gehört das Löschen und Ändern der Datensätze, aber auch das Verifizieren dieser.

Diese Informationen werden, im Vergleich zu den Artinformationen nur zum Teil öffentlich dargestellt. Zusätzlich sind diese Informationen mit der im Zusammenhang mit der Art erwähnten Karte verknüpft, weshalb die weitere Beschreibung im folgenden Bereich über die geographischen Funktionen vorgenommen wird.

Neben den beiden vorhergehend beschriebenen Aufgabenbereichen, sollen registrierte und administrative Nutzer auch die Möglichkeit haben Bilder zu den biologischen Arten hochzuladen. Diese Aufgabe muss aber nicht zwingend über die, durch diese Arbeit entstehende Realisierung abgedeckt werden.

2.1.2 Geographische Informationen

Neben der Erfassung der allgemeinen Informationen ist die Erfassung und Darstellung von geographischen Informationen Teil dieser Arbeit. Für *GIS* im Kontext der *Biodiversität* sind hierfür zwei Aufgabenbereiche zu unterscheiden. Zum einen geht es um die im vorhergehenden Teil angesprochene Erfassung und Darstellung von Orten, an denen Vorkommen von *Biodiversitäten* gefunden worden sind. Zum anderen geht es darum den Bezug dieser Vorkommen zu einer Karte herzustellen, da Koordinaten alleine nur wenig Aussagekraft besitzen.

In den Funktionen für allgemeine Informationen wurde schon angedeutet, dass eine Art über eine Liste ausgewählt werden kann und daraufhin mit weiteren Informationen und einer Karte angezeigt werden soll. Die Karte soll dazu dienen, durch Nutzer eingetragene Vorkommen zu visualisieren und ist für jeden Nutzertyp sichtbar.

In der ursprünglichen Version gab es eine statische Karte, auf der die eingetragenen Punkte dargestellt worden sind. Durch Überfahren einzelner Punkte mit der Maus, konnten zusätzlich Informationen angezeigt werden. Die Punkte wurden immer ungenau dargestellt, da sie Mittelpunkte von länderspezifischen Rastereinteilungen darstellten. Diese Form der Kartendarstellung wurde so nicht übernommen, es sind jedoch Anlehnungen daran zu finden.

In der neuen Umsetzung ist eine dynamische Karte gefordert. Dabei ist gewünscht das diese Karte, die zur Darstellung der Vorkommen einer Art genutzt wird, einen möglichst großen Platz bekommt, da sie der wichtigste Teil eines Atlas sein sollte. Zusätzlich soll die Karte mindestens mit einer maximalen Zoomstufe von 1:100.000 ausgestattet sein. Sollten höhere Zoomstufen möglich sein, können diese, falls gewünscht, verwendet werden. Die Entscheidung darüber liegt bei den administrativen Nutzern.

Beim ersten Aufruf einer Seite soll eine Karte mit einer im Vorfeld durch die administrativen Nutzer festgelegten Zoomstufe gezeigt werden. Diese Zoomstufe soll variabel sein und sich an dem gewünschten Gebiet orientieren. Für eine Übersicht über die eingetragenen Fundorte im Fall Deutschland würde die Zoomstufe zum Beispiel so gewählt werden, dass die Grenzen Deutschlands gerade noch im Sichtbereich der Karte erscheinen würden. Es soll jedoch nicht nur eine Zoomstufe geben. Der Nutzer soll die Karteninhalte selbstständig vergrößern oder verkleinern können. Bei der weiteren Navigation zwischen verschiedenen Arten soll die von dem Nutzer gewählte Zoomstufe anstatt der voreingestellten Zoomstufe verwendet werden. Die Zoomstufen müssen nicht zwingend gleitend sein, exemplarisch sind Zoomstufen für Deutschland und Österreich wie folgt sinnvoll:

- Beide Länder

- Einzelne Länder
- Bundesländer

Neben einer Zoomfunktion soll sich der Nutzer in der Karte bewegen können. Um den Fokus auf ein durch die Betreiber der Seite gewünschtes Gebiet nicht zu verlieren, soll die Bewegungsfreiheit allerdings begrenzbar sein, diese Einschränkung der Sicht ist nur der Gruppe der administrativen Nutzer vorbehalten.

Es soll ermöglicht werden die dargestellte Karte mit weiteren Informationen auszustatten. Die Karte eines *GIS* setzt sich normalerweise aus verschiedenen auswählbaren Ebenen zusammen. Diese Ebenen, im Englischen auch Layer genannt, werden häufig dazu genutzt, eine Karte Schritt für Schritt mit Informationen ergänzen zu können. Es ist jedoch auch möglich, um eine überfordernde Informationsflut zu vermeiden, einzelne Ebenen auszublenden. Dieses Prinzip soll auch in dieser Anwendung genutzt werden, um beispielsweise andere Gebietszuordnungen zeigen zu können. Am Beispiel der Fische wären diese unter anderem:

- Flusseinzugsgebiete
- Schutzgebiete gem. NATURA 2000

Damit Ebenen verfügbar sind, muss es eine Möglichkeit geben, mit der, Ebenen eingefügt und entfernt werden können. Bei den daraus entstehenden Ebenen ist es nötig eine Auswahl treffen zu können, welche Ebenen für eine Kartendarstellung verwendet werden sollen und welche nicht. Die Einbindung von Ebenen, und damit die Flexibilität der Kartendarstellung sollte nicht auf fixierten Quellen beruhen. Die Nutzbarkeit der Daten wäre sonst erheblich eingeschränkt. Daraus ergibt sich, dass langfristig keine Abhängigkeit zu den jeweiligen Kartendatenanbietern und den zum Teil vorhandenen *Webservices* bestehen darf. Verschiedene Quellen sind für die geforderten Ebenen nötig, daher wäre eine Umsetzung mit einer solchen Abhängigkeit unmöglich. Die Verwaltung von Karteneigenschaften ist allerdings nur den administrativen Nutzern vorbehalten, alle weiteren Nutzer haben nur die Möglichkeit, die, durch die administrativen Nutzer, freigegebenen Ebenen ein- oder auszublenden.

Neben den generellen Eigenschaften der Karte gibt es, wie schon erwähnt, Funktionen, die sich auf die Darstellung und Erfassung von Fundorten beziehen. Grundsätzlich sollen Fundorte als Punkte auf der Karte dargestellt werden, sollten mehrere Fundorte an einem Punkt vorhanden sein, muss dies verdeutlicht werden. Zusätzlich soll es drei verschiedene Qualitäten für Punkte geben. Die Qualität eines Punktes soll anhand seiner Darstellung in der Karte erkennbar sein und es soll registrierten Nutzern möglich sein, eigens eingetragene Vorkommen zu erkennen. Folgende Qualitäten sind vorhanden:

- **Hystorisch** – Vorkommen mit einem Datum vor 1980.

- **Ungenau** – Vorkommen ohne Datum oder mit als ungenau gekennzeichneten Koordinaten.
- **Genau** – Alle weiteren Vorkommen mit genauen Koordinaten.

Die Datumsgrenze 1980 ist vom Projektteam bestimmt worden, da mit dem Zeitraum um das Jahr 1980 ein aktiverer Umweltschutz als zuvor begann.

Da Umweltschutz ein Thema des Atlanten ist, ist der Fragestellung nach dem bestmöglichen Schutz der Tiere in Verbindung mit der Offenlegung ihrer Vorkommen nachzugehen. Gerade gefährdete Arten werden einem erhöhten Risiko ausgesetzt, sofern allgemeine Internetnutzer, als anonyme Personen, die Vorkommen einsehen können. Aus diesem Grund dürfen solche Daten nur den Nutzern zugänglich gemacht werden, die auch verantwortlich damit umgehen.

Aus diesem Grund ist es gewünscht, zwei Ansichten der Fundorte zur Verfügung zu stellen. Eine verfälschte Darstellung der Daten für den Gast und eine Möglichkeit für den registrierten Nutzer genaue Daten einzusehen. Im Fall Deutschland bedeuten verfälschte Daten beispielsweise das Koordinaten als Rasterkoordinaten in Form von *TK25-Mittelpunkten* angezeigt werden, welche keine genaue Zuordnung eines Punktes zulassen. Diese Form der Darstellung entspricht der derzeitigen Darstellung in der ersten Umsetzung. Als genaue Koordinaten sollen Koordinaten des *WGS84*-Weltkoordinatensystems eingesetzt werden, um die Anwendung auch international verwenden zu können.

Eine letzte Anforderung an die Karte ist die Möglichkeit, Informationen zu einzelnen Fundorten entweder per Klick oder durch überfahren mit der Maus abzurufen. Die Fundortinformationen für registrierte Nutzer sollen sich aus Datenquelle, Nachweisdatum und falls durch den Finder gewünscht, einem Kommentar zusammensetzen. In der verfremdeten Darstellung für Gäste soll eine Liste der in einem Punkt vorhandenen Vorkommen angezeigt werden. Diese Liste zeigt nur die Quelle der einzelnen Vorkommen.

Für die Erfassung der Vorkommen ist ebenfalls eine Karte nötig. Diese Karte muss jedoch nicht die vorausgehend erläuterten Kriterien erfüllen, sie soll nur zur Vorschau und Überprüfung eines Punktes und dessen Lage dienen. Um ein Vorkommen einzutragen, müssen für dieses Vorkommen Koordinaten in Form von Längen- und Breitenangaben vorhanden sein. Sind diese Koordinaten aus dem *WGS84*-System entnommen, soll ein Nutzer vor dem Speichern durch eine Vorschau die Korrektheit seiner Eingaben überprüfen dürfen. Da es jedoch nicht immer der Fall ist, dass Koordinaten in Form von *WGS84*-Koordinaten vorliegen, soll eine Transformationsfunktion für diverse Koordinatensysteme zum *WGS84*-Koordinatensystem eingebunden werden.
Die Transformation von Koordinaten gehört zu diesen Anforderungen, da die Durchführung einer Koordinatentransformation nicht trivial ist, ein einheitliches

Format für eine konsistente Speicherung und Darstellung aber nötig ist. Nur so kann sichergestellt werden, dass verschiedene Koordinatensysteme mit geringen Abweichungen in das *WGS84*-Format gemäß *EPSG Code* 4326 überführt werden können. Auch in diesem Fall soll nach der Transformation eine Vorschau zur Überprüfung der Angaben möglich sein.

Da es passieren kann, das Vorkommen nur den administrativen Nutzern zur Verfügung gestellt werden sollen, ist eine Funktion gewünscht die jedem Nutzer, der berechtigt ist, Vorkommen einzutragen, die Option zur Verfügung stellt seine Daten nur für sich und die Nutzer des Types „administrativ“ freizugeben. Diese Daten dürfen jedem anderen Nutzer nur verfremdet dargestellt werden.

2.2 Technische Anforderungen

Anhand der Funktionen können zusätzlich technische Anforderungen abgeleitet werden. Diese Anforderungen sind wie bereits gesagt nur für die Implementierung wichtig und werden daher auch nicht nach Nutzertypen unterschieden.

2.2.1 Allgemeine Informationen

Da Joomla! als *Content Management System* für die Verwaltung einfacher Webinhalte ausgelegt ist, fehlt die Möglichkeit die unter den Funktionen angesprochenen Inhalte systematisch zu erfassen. Aus diesem Grund ist es nötig Joomla! um diese Funktionalität zu erweitern. In Anlehnung an das ursprüngliche System sollte es ebenfalls mit Hilfe einer eigens dafür entwickelten Komponente durchgeführt werden. Nur mit Hilfe einer solchen Komponente ist es möglich, zukünftig aufbauend auf einer Joomla!-Grundinstallation einen Atlas zu erzeugen und zu erweitern.

Da die zu realisierende Anwendung die Defizite der ursprünglichen Anwendung ausgleichen soll, ist ein neues Datenmodell nötig. Auch wenn einige Teile der alten Anwendung übernommen wurden, gibt es dennoch Neuerung und das Bestreben eine Anwendung zu realisieren, die möglichst generisch ist. Das bedeutet, es ist ein Datenmodell gefordert, das zum einen kompatibel zu den alten Modellen ist, um Daten migrieren zu können und zum anderen eine Möglichkeit bietet, möglichst verschiedene Biodiversitätsdaten zu erfassen. Da Joomla! eine eigene Datenstruktur, besitzt und diese Arbeit einen direkten Bezug zu Joomla! hat, soll das neue Datenmodell außerhalb dieser Arbeit entworfen werden. So wird sicher gestellt, dass das neue Datenmodell nicht auf Basis der Joomla!-Datenstruktur realisiert wird. Hierdurch wird versucht, die Daten generisch zu erfassen und nicht von Joomla! abhängig zu machen. Nur durch einen generischen Ansatz der *Persistenz* ist es zum Beispiel möglich, zukünftig einen *Mapserver* für weitere *GIS*-Funktionen einzubinden.

Um das System, sei es Datenhaltung oder Webanwendung, später erweitern zu können soll in dieser Arbeit vor allem die Verwaltung und Darstellung der Kerndaten umgesetzt werden. Zukünftige Entwicklungen sollen auf die dadurch entstehende Kernanwendung aufbauen können.

2.2.2 Geographische Informationen

Für die *GIS*-Funktionen ist es vor allem wichtig auf vorhandenes Wissen zurück zugreifen. Obwohl sämtliche Funktionen durch eigene Lösungen realisiert werden könnten, ist es ausdrücklich gewünscht, dies nicht zu tun. Es gibt inzwischen viele Standards im *GIS*-Bereich und zusätzlich Bibliotheken, die diese Standards korrekt implementieren. Die Verwendung solcher Bibliotheken stellt sicher, dass die Anwendung zukünftig einsetzbar bleibt, und ist daher Teil der Anforderung nach einer generischen Anwendung.

2.2.3 Software Vorgaben

Da die Webanwendung derzeit auf Joomla!-basiert, ist es gewünscht Joomla! weiterhin als *CMS* zu nutzen. Dieser Wunsch resultiert nicht zuletzt aus dem leichten Einstieg in den Umgang mit diesem System. Sollte also ein anderes System nötig sein, muss es zumindest einen leichten Einstieg bieten und einfach in der Handhabung sein. Die Verwendung von Joomla! impliziert allerdings, dass die Datenspeicherung in einer MySQL Datenbank vorgenommen werden muss, da Joomla! derzeit nur mit MySQL kompatibel ist. In der folgenden Analyse wird hierauf noch einmal genauer eingegangen.

Diese Tatsache steht jedoch in Konkurrenz mit dem zweiten Wunsch der derzeitigen Nutzer, aus Gründen der besseren Auswertbarkeit von Geodaten, das Datenbankmanagementsystem PostgreSQL mit seiner Erweiterung PostGIS einzusetzen. Die Diskussion über eine Lösung der konkurrierenden Wünsche ist Teil der Werkzeuganalyse und der Konzepte. Die Verwendung weiterer Software ist dem Entwickler freigestellt.

2.3 Begrenzungen

Da es durchaus möglich ist eine Vielzahl von Funktionen mit einem *GIS* bereitzustellen, sollen im Folgenden einige Einschränkungen vorgenommen werden.

Neben der Darstellung einer Karte mit möglichst weitreichenden Informationen, sowie der Erfassung und Verwaltung von einzelnen Daten, ist eine zukünftige Perspektive die Erfassung von Massendaten. Diese Erweiterung ist derzeit jedoch nebensächlich, da größere Datenmengen bereits eingepflegt sind und der Atlas

durch individuelle Daten erweitert werden soll, die in der Menge typischerweise geringer sind als, die Menge der Daten von einem Institut.

Neben dem Import, ist auch die Möglichkeit detaillierte Abfragen zum Export von Daten zu machen vorerst nicht Teil dieser Arbeit.

Eine weitere Einschränkung besteht in der Möglichkeit von Nutzerabfragen zu den in der Karte dargestellten Daten, z.B. Mengenverhältnisse und Flächendaten. Da einige Abfragen die Komplexität eines *Mapservers* benötigen, wird auf solche Abfragen vorerst verzichtet, sie sind damit nicht Teil dieser Arbeit, da die damit entstehenden Kosten zurzeit nicht tragbar wären.

Thema dieser Arbeit ist, die Reimplementierung und Erweiterung der ursprünglichen Umsetzung um ein generisches System zu erzeugen, das sich in Joomla! integriert und mehrfach einsetzbar ist, alles darüberhinausgehende, ist nicht im Fokus.

3 Aktuelle Methoden und Technologien

Um Daten im Web darzustellen, gibt es diverse Werkzeuge und Möglichkeiten. Diese reichen von einer eigenständigen Realisierung ohne Hilfsmittel, deren Wartung entsprechend aufwändig ist, über Umsetzungen mit Hilfe von Frameworks bis zur Anpassung existierender Webanwendungen wie beispielsweise *Content Management Systeme*. Da, die zu entwickelnde Webanwendung, neben der Darstellung einfacher Text- und Bild-Informationen auch mit komplexeren Geodaten arbeiten muss, weichen die Anforderungen jedoch von denen für eine einfache Website mit reinen Text- und Bildinformationen ab. Streng genommen sind graphisch dargestellte Geodaten in einer Website zwar ebenfalls als Bilder vorhanden, aufgrund des interaktiven Charakters der Geodarstellungen in diesem Kontext werden sie jedoch gesondert betrachtet. Dies ist nötig, da sie zusätzliche Werkzeuge erfordern, die den Umgang mit Geodaten unterstützen. Hierzu zählen Werkzeuge für die Darstellung, Speicherung und Verarbeitung der Geodaten. Im Folgenden soll eine Auswahl an möglichen Werkzeugen zur Realisierung der, dieser Arbeit zugrundeliegenden, Webanwendung diskutiert werden. Dabei wird unterschieden in Werkzeuge für die Darstellung im Web, für die Speicherung der Daten und für die Verarbeitung von Geoinformationen.

3.1 Webdarstellung

Auch wenn die Anwendung mit Geodaten arbeitet, stellt sie auch einfachere Informationen, wie zum Beispiel die Beschreibung einer biologischen Art oder Informationen über das Projekt dar. Diesbezüglich wurde schon in den Anforderungen erwähnt, dass eine leichte Verwaltung dieser Daten und der Geodaten möglich sein muss. Es wäre möglich eine komplett neue Anwendung zu entwickeln, die den Anforderungen gerecht wird. Dieser Ansatz ist jedoch nicht effizient, da viele Lösungen vorhanden sind, die die Verwaltung von Informationen möglich machen.

Gemeint ist ein mindestens *Web-basiertes Content Management System* (*WCMS*/ *CMS*). In den meisten *CMS* wird zusätzlich auf eine Möglichkeit der gemeinsamen Bearbeitung von Inhalten wert gelegt.[Ale09] Es gibt derzeit viele verschiedene *CMS* im Internet, einige sind sich in ihrer Funktionalität sehr ähnlich, während wiederum andere sehr verschieden sind. Es gibt spezielle *WCMS* die auf bestimmte Anwendungsgebiete, wie zum Beispiel Blogs oder Wikis zugeschnitten sind und

allgemeine Systeme für die Darstellung verschiedenartiger Websites.[Ale09]

Da das ursprüngliche System schon zum gewissen Teil in das *WCMS* Joomla! eingebettet ist, wird dieses *WCMS* vorrangig betrachtet. Vor allem weil Joomla! und dessen Handhabung dem Projektteam schon bekannt sind und das Projektteam daher für die Nutzung dieses *WCMS*. stellvertretend für die Nutzer, eintritt, bietet sich eine Umsetzung in Verbindung mit diesem System an. Es besteht dadurch zusätzlich die Möglichkeit, vorhandene Strukturen wiederzuverwenden und in das Resultat dieser Arbeit einfließen zu lassen. Der durch einen Wechsel des Systems entstehende Aufwand sollte ebenfalls nicht außer Acht gelassen werden. Auch wenn die Daten, wie für ein *CMS* üblich, nicht mit der Darstellung verknüpft sind, sind sie dennoch in der von Joomla! vorgegebenen Struktur abgelegt, wodurch eine Portierung der Daten und die Einarbeitung des Projektteams in das neue System, unumgänglich wären. Aus diesem Grund werden andere Systeme nur dann im Rahmen dieser Arbeit betrachtet, wenn der Einsatz von Joomla! in der folgenden Untersuchung besonders schwer, beziehungsweise nicht möglich ist.

3.1.1 Joomla

Joomla! ist in erster Linie als *CMS* für den Einsatz im Web konzipiert und damit ein *WCMS*.[Ale09] Daraus resultiert die für Webanwendungen typische, benötigte Konstellation aus einem Webserver und einer Datenbank.

Der große Vorteil von Joomla! im Vergleich zu anderen *CMS* ist die simple Handhabung durch den Nutzer. Durch eine strikte Trennung von Verwaltung und Anzeige in Frontend und Backend sind Funktionen eindeutig zugeordnet. Im Frontend wird eine Website dargestellt und dem Nutzer eine begrenzte Anzahl an Interaktionsmöglichkeiten gegeben, während im Backend die Anzeige sowie der Inhalt der Seite konzipiert und gestaltet werden kann. Da Frontend und Backend auf dem Webserver liegen, sind keine weiteren Werkzeuge zur Verwaltung notwendig.[Ale09, JOO10]

Das System Joomla! basiert auf einem eigenen Framework, das mit PHP umgesetzt ist.[Ale09, JOO10] Da PHP bei den meisten Hosting-Anbietern beispielsweise als Linux, Apache, MySQL und PHP (*LAMP*) Zusammenstellung verfügbar ist, ist Joomla! auch auf den meisten Hosting-Angeboten einsetzbar.

Auf Basis des Frameworks besitzt Joomla! eine Struktur aus Komponenten, Modulen und Plugins, die zusammen das Joomla!-*WCMS* bilden. Durch die Modularität wird ein Redaktionssystem geschaffen, das dem Nutzer verschiedene, jeweils spezialisierte Erweiterungen zur Verwaltung seiner Website zugänglich macht und damit die Verwaltung wesentlich vereinfacht. In der Grundinstallation wird Wert auf die typischen Inhalte einer Webseite gelegt, beispielsweise Medienmanager, Artikelverwaltung, Verwaltung für Kontaktdaten, sind neben weiteren Funktio-

nen für die Darstellung der Webseite vorhanden. Der Umfang der angebotenen Funktionen ist allerdings nicht festgelegt. Durch die in Abbildung 3.1 dargestellte Framework-Struktur ist es möglich eigene Erweiterungen für neue oder komfortablere Funktionen zu entwickeln, die direkt in das *CMS* eingebettet werden können. Wie in Abbildung 3.1 erkennbar, setzt sich eine Anwendung in Joomla! im wesentlichen aus drei Elementen zusammen.[Ale09]

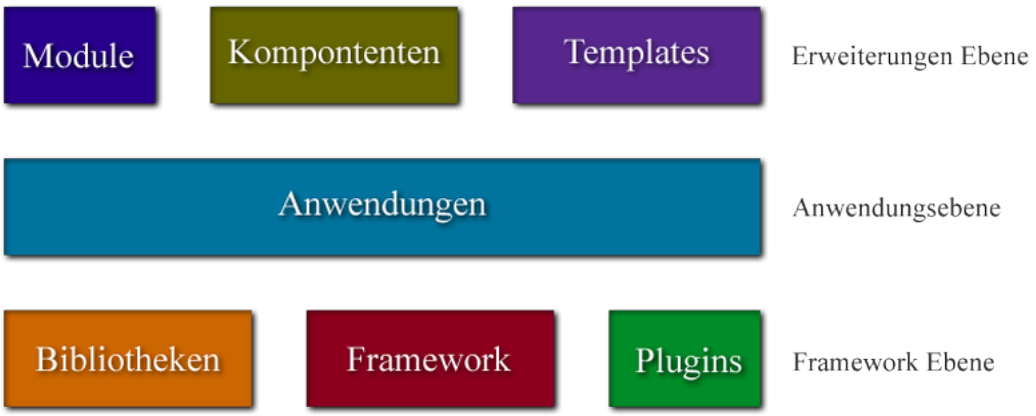

Abbildung 3.1: Joomla!-Architektur
Quelle: http://wiki.joomla-nafu.de/joomla-dokumentation/Datei:800px-Joomla_Architektur.png

Aus Tabelle 3.1 ist zu erkennen, dass für die verschiedenen Anwendungsfälle klare Strukturen definiert sind. Diese Strukturen sowie verschiedene vordefinierte Funktionen können den Entwicklungsaufwand deutlich verringern.

Um die Entwicklung weiter zu vereinfachen, wird versucht Joomla!, seit Version 1.5, strikt auf Basis des *Model-View-Controller(MVC)-Musters* weiterzuentwickeln. Dadurch wird die Darstellung von der Geschäftslogik und dem Speicher getrennt. So sind für dieselben Daten unterschiedliche Ansichten und Operationen möglich.[Ale09, JOO10] Abbildung 3.2 zeigt das *MVC-Musters* in Form eines Modells. Abweichend von der Abbildung muss nicht immer genau ein Model mit genau einer View und genau einem Controller verbunden sein. Aus Gründen der Übersicht wird meistens ein Model mit einer dazugehörigen View verknüpft. Ein Controller kann dagegen verschiedene Models und Views ansprechen und zueinander zuordnen. Durch eine eigenständige Speicherschicht ist das *CMS* ebenfalls in der Lage verschiedene Speicherarten zu nutzen, es können unter anderem Datenbanken oder Dateisysteme angesprochen werden. Als Datenbank kommt bei Joomla! derzeit aber nur MySQL in Frage, langfristig ist jedoch eine Unabhängigkeit in Bezug auf das Datenbankmanagementsystem geplant. [Ale09, JOO10]

Element	Einsatz
Komponente	Integration von neuen Funktionen und benötigten Verwaltungsstrukturen, meist mit eigenen Tabellen in der Datenbank.
Modul	Darstellung von Informationen an vorher bestimmten Stellen. Ein Modul greift üblicherweise auf vorhandene Datenbankeinträge zu.
Plugin	Bereitstellung von Ereignissen zum Eingriff in die Ablaufsteuerung des Frameworks, um vorhandene Strukturen an verschiedenen Stellen zu ergänzen.

Tabelle 3.1: Joomla! Anwendungselemente
Quelle: [Ale09, Anj10]

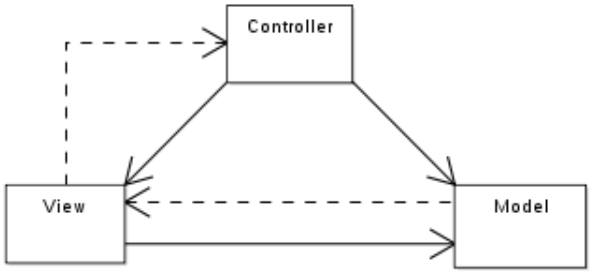

Abbildung 3.2: MVC-Muster
Quelle: http://wiki.joomla-nafu.de/joomla-dokumentation/Datei:321px-ModelViewControllerDiagram.png

Für die Webseitendarstellung, gibt es *Templates*, die angepasst oder selbst entwickelt werden können. Besonders interessant ist das Beez-*Template*, das einen barrierefreien Ansatz verfolgt. Jede Erweiterung besitzt durch das *MVC-Muster* ebenfalls *Templates* zur Darstellung der eigenen Inhalte in der Webseite. Um trotzdem ein einheitliches Design zu ermöglichen, können die *Templates* der Erweiterungen im globalen Webseiten-*Template* überschrieben und die Erweiterungen anderer Entwickler an das eigene Design angepasst werden. Durch die Nutzung von Sprachdateien kann Joomla! auch international eingesetzt werden.[Ale09, JOO10]

Für die Zugriffssteuerung bietet Joomla! ein vordefiniertes Gruppenmodell an, in das Nutzer eingeteilt werden können. Es wird hierbei ebenfalls in Frontend und Backend unterschieden. Die Tabelle 3.2 stellt einen kurzen Überblick der verfügbaren Gruppen dar. Mit der Definition von Access Controll Lists bei der Entwicklung einer Erweiterung kann gezielt bestimmt werden welche Gruppen Zugriff auf die Einstellungen und Daten einer Komponente haben.[Anj10]

Bezeichnung	**Zugang**	**Rechte**
Guest	Frontend	öffentliche Bereiche ansehen
Registered	Frontend	interne Bereiche ansehen
Author	Frontend	Inhalte erzeugen eigene Inhalte bearbeiten
Editor	Frontend	sämtliche Inhalte bearbeiten
Publisher	Frontend	sämtliche Inhalte veröffentlichen
Manager	Backend	Menü verwalten Sections erzeugen Sections bearbeiten Categories erzeugen Categories bearbeiten Media Manager nutzen Statistiken nutzen
Administrator	Backend	Komponenten verwalten Module verwalten Plugins verwalten Benutzer verwalten Nutzungsrechte bis Administrator vergeben
Super Administrator	Backend	*Templates* verwalten Systemnachrichten empfangen Systeminformationen abfragen Systeminformationen bearbeiten

Tabelle 3.2: Joomla! Nutzergruppen
Quelle:[Anj10]

Da Joomla! unter der GNU – General Public License (GNU-GPL) steht, zählt es als Open Source und ist kostenlos zugänglich, frei veränderbar und uneingeschränkt einsetzbar. Jedoch besteht durch diese Offenheit auch keinerlei Gewährleistung, [JOO10, Ale09] vorausgesetzt es gibt kein gesondertes Vertragsverhältnis. Dank des Open Source Charakters, existiert aber eine große Community, die sich mit der Entwicklung und Weiterentwicklung des Systems befasst. Ein Core Team steuert und überwacht diese Entwicklung. Neben den Entwicklern von Joomla! gibt es zusätzlich noch viele weitere Entwickler, die mit Hilfe des angesprochenen Frameworks, Erweiterungen für Joomla! implementieren. Daraus resultieren viele Einsatzmöglichkeiten, bei denen eigene Entwicklungsarbeiten gegebenenfalls entfallen können.[Ale09]

Open Source hat jedoch auch eine Schattenseite, durch den offenen Quellcode ist es jedem möglich, Sicherheitslücken im *CMS* zu suchen und auszunutzen. Sicherheitslücken entstehen allerdings meist nicht durch den Joomla!-Kern, sondern durch Fehler in der Entwicklung von Erweiterungen. Prinzipiell stellt Joomla! eine Vielzahl an Funktionen zur Verfügung, um den typischen Bedrohungen im Web wie *SQL-Injection* oder *Cross-Site-Scripting* entgegen zu wirken. Maskieren von SQL-Abfragen und das gezielte Formatieren von Request-Parametern sind Beispiele dafür.[Ale09]

Da Joomla! ein *WCMS* ist, bietet es in der Grundinstallation keine Unterstützung zur Verarbeitung von Geodaten, das *CMS* lässt sich aber durch den angesprochenen Aufbau auf den Umgang mit Geodaten anpassen. Das dies funktioniert, zeigt eine kostenpflichtige Erweiterung mit dem Namen WISroGIS[1]. Diese Erweiterung ermöglicht, das Einbinden verschiedener Karten von diversen *Webservices* und bietet die Möglichkeit der Darstellung eigener Strukturen mit Hilfe der Keyhole Markup Language(KML). Die Darstellung der Geodaten wird durch die JavaScript-Bibliothek OpenLayers ermöglicht. [WIS30] WISroGIS wurde jedoch für die Lösung dieser Arbeit aufgrund der Kosten nicht in Betracht gezogen.

Prinzipiell ist kein Wechsel auf ein anderes *CMS* nötig. Eine genaue Erläuterung weiterer Gründe gegen einen Wechsel ist in den Folgerungen zu finden.

3.1.2 OpenLayers und GeoExt

Es gibt vielfältige Wege Kartendaten zu nutzen. Im Internet gibt es verschiedene *Webservices* die Kartendaten zur Verfügung stellen, einige werden im weiteren Verlauf noch einmal genauer beschrieben. *Webservices* haben meist eine eigene API zur Abfrage von einzelnen Geodaten oder ganzen Karten. Diese API kann gemäß eines, vom *Open Geospatial Consortium*(*OGC*) im Standard *OpenGIS Web Services*(*OWS*) definierten, *Webservice* umgesetzt sein. Sie kann aber auch abweichend oder gänzlich verschieden sein. Hinzu kommt die Möglichkeit, auf Daten aus dem Internet zu verzichten und Daten lokal auf dem eigenen Web- oder *Mapserver* vorzuhalten. Jede Methode hat ihre eigenen Voraussetzungen, auf die bei einer Einbindung geachtet werden muss.

OpenLayers ist eine JavaScript-Bibliothek zur serverunabhängigen Darstellung von Geodaten.[OLD10] OpenLayers bietet eine von den unterschiedlichen Voraussetzungen unabhängige API, die mit Hilfe einer Abstraktionsschicht und *Asynchronous JavaScript and FML*(*AJAX*), verschiedene Datenquellen verwenden kann, dabei ist ferner die Möglichkeit gegeben, mehrere Datenquellen ergänzend mitein-

[1] http://www.wis.ro/index.php/en_GB/Software-Development/Joomla-Extensions/View-all-products.html

ander zu verbinden.[Mar10]

Für die Interaktion mit der Karte bietet OpenLayers verschiedene Navigationswerkzeuge, deren Funktionsweise schon von anderen Kartendiensten wie Google Maps bekannt ist. Eine Zoomfunktion und eine Minikarte zur Navigation sind zum Beispiel vorhanden, das Verschieben der Karte im Sichtfenster ist ebenfalls möglich. Durch Ebenen, auch Layer genannt, können verschiedene Kartendarstellungen an- und abgewählt und dadurch kombiniert werden. OpenLayers kann diesbezüglich verschiedene, im *GIS*-Umfeld typische, Formate lesen. Eine Liste der Formate ist in der Dokumentation zu OpenLayers zu finden[2].[OLD10]

Besonders interessant ist die von OpenLayers angebotene Funktion *Vektordaten* im Client zu rendern und deren Darstellung manipulieren zu können. Dies ermöglicht zum Beispiel die graphische Darstellung von Daten im *Geography Markup Language*(*GML*) Format. Die Umsetzung dieser Funktion, ist einer *XML-Transformation*(*XSLT*) mit der *Extensible Stylesheet Language*(*XSL*) sehr ähnlich, den einzelnen Vektorelementen werden dabei Darstellungseigenschaften zugewiesen. Aufgrund der Tatsache, dass die Daten durch den Browser gezeichnet werden müssen, ist natürlich ein Performancelimit bezogen auf die Menge der zu rendernden Daten vorhanden.[Mar10]

Intern basiert OpenLayers, neben den eigens dafür entwickelten Funktionen, auf einem Teil der JavaScript-Bibliothek Protoype JS[3]. Dieser Teil dient zur Navigation im *Document Objekt Model*(*DOM*) und zur Selektion der enthaltenen Elemente. Für die asynchrone Kommunikation wird XMLHttpRequest.js[4] als *AJAX*-Schnittstelle verwendet. Diese Schnittstelle sorgt für die nötige Kompatibilität mit den verschiedenen Browsern. Für Teile der Darstellung wird die JavaScript-Bibliothek Rico[5] verwendet. Mit Rico können einfache Web2.0 - Effekt erzeugt werden. Ein weiterer Bestandteil ist Gears[6], eine JavaScript-Bibliothek, die Interaktionen mit lokalen Elementen außerhalb des Browsers ermöglicht.[Mar10]

OpenLayers ist wie Joomla! Open Source, es besitzt jedoch eine BSD-artige Lizenz, dies bedeutet, ähnlich der Lizenz von Joomla! sind Nutzung und Veränderungen der Quellen erlaubt. Der Unterschied besteht hauptsächlich darin, dass kein Copyleft vorhanden ist und die Quellen nicht zwingend verfügbar sein müssen. Copyleft bedeutet, ein von OpenLayers abgeleitetes Produkt muss nicht zwingend, unter derselben BSD-artigen Lizenz stehen.[Mar10]

Trotz der vielen Bibliotheken die in OpenLayers eingebunden sind und der Fle-

[2]http://docs.openlayers.org/library/index.html
[3]http://www.prototypejs.org/
[4]http://code.google.com/p/xmlhttprequest/
[5]http://openrico.org/
[6]http://tools.google.com/gears/

xibilität in der Wahl der Kartenquellen, ist OpenLayers, vor allem durch das objektorientierte Konzept der API und der Möglichkeit die Bibliothek auf wenige Dateien zu reduzieren, in jede Website importierbar und anwendbar. Dabei muss jedoch vorausgesetzt werden, dass der Client JavaScript unterstützt.

Das Konzept von OpenLayers wird durch die Tatsache bestätigt, dass ein ähnlicher Ansatz mit dem Namen Mapbuilder[7] zugunsten von OpenLayers eingestellt wurde.[Mar10]

GeoExt

GeoExt ist eine JavaScript-basierte Werkzeugsammlung zur Erzeugung von Rich Web Mapping Anwendungen. Dazu werden die Funktionen der OpenLayers-Bibliothek für die Nutzung von Geodaten mit den Funktionen des Ext JS[8] JavaScript-Frameworks zur einfachen Erzeugung desktopähnlicher *Graphical User Interface*(*GUI*)-Komponenten kombiniert. GeoExt stellt damit eine API zur Verfügung, die es ermöglicht die Inhalte von OpenLayers komfortabel in *GUI*-Komponenten einzubetten. Es erweitert damit das in OpenLayers durch die Bibliothek Rico vorhandene Spektrum um desktopähnliche Web2.0 Effekte aus dem Ext JS-Framework. GeoExt steht wie OpenLayers unter einer BSD-artigen Lizenz.[GEX05]

3.2 Datenverwaltung

Wie schon bei der Charakterisierung von Joomla! erwähnt, benötigt Joomla! MySQL als Datenspeicher, daher soll MySQL zunächst betrachtet werden. Im Zusammenhang mit Geodaten gibt es weitere Möglichkeiten, diese werden im Folgenden ebenfalls betrachtet, um einen Überblick über mögliche Lösungsansätze zu geben.

3.2.1 MySQL

MySQL wird von den Vertreibern, als das populärste Open Source Datenbanksystem der Welt bezeichnet.[MYS08] Gestärkt wird diese Aussage durch die Tatsache, dass MySQL im Web erfahrungsgemäß sehr häufig vertreten ist und von den meisten Hosting-Anbietern zur Verfügung gestellt wird.

MySQL ist ein relationales Datenbanksystem, dessen Architektur aus dem Datenbankmanagement und verschiedenen Datenbanken besteht. Prinzipiell lassen sich in MySQL die für Datenbanksysteme üblichen Funktionen finden. Es gibt die Möglichkeit von Schreib- und Leseoperationen, zur Abfrage gibt es zusätzlich verschiedene Strukturen wie Joins und Where-Klauseln um Daten miteinander zu

[7] http://communitymapbuilder.org/
[8] http://www.sencha.com/products/js/

verknüpfen und gegeneinander zu selektieren. Zum Schutz der Daten ist eine Verwaltung vorhanden, um Nutzern spezifische Rechte zuzuweisen und mit Hilfe von sogenannten Views die Sichtweise auf eine Datenbank einzuschränken. Um Metadaten der jeweiligen Datenbankstruktur abzufragen, wird ein Informationsschema vorgehalten.[MYS08]

Die Besonderheit von MySQL liegt in dem Datenbankmanagement. Für das Management gibt es verschiedene, eigenständige Engines. Jede Engine kann als alleinstehendes Management eingesetzt werden. Die bekanntesten Engines sind MyISAM und InnoDB. Beide Engines, unterstützen unterschiedliche Möglichkeiten zum Datenmanagement. Während InnoDB eine Möglichkeit besitzt, Transaktionen, also Lese- und Schreibzugriffe, Datenbank-weit zu steuern und Referenzen zwischen Tabellen zu verwalten, wird bei MyISAM aus Performanzgründen auf einen Teil dieser Funktionen verzichtet. Im Vergleich zu MyISAM bietet InnoDB also einen besseren Schutz vor inkonsistenten Daten, durch gleichzeitige Schreibzugriffe und die Möglichkeit Fremdschlüssel und die damit verknüpften Bedingungen zu verwalten. MyISAM hat dafür eine höhere *Performanz*, da es weniger Abhängigkeiten bei einem Datenzugriff überwacht. [MYS08]

MySQL kann durch die Unterstützung diverser Treiber in den meisten Programmiersprachen eingesetzt werden. Es gibt unter anderem für die Programmiersprache Java einen JDBC-Treiber, der die Anbindung einer MySQL-Datenbank an ein Javaprogramm ermöglicht.[MYS08]

Ein Vorteil, jedoch auch ein Nachteil von MySQL liegt in der *Performanz*. Bei der Entwicklung von MySQL wird großen Wert auf *Performanz* gelegt, dabei wird jedoch die Konformität mit SQL-Standards vernachlässigt. Dies ist unter anderem in Prozeduren zur einfacheren Handhabung von Daten zu finden.[MYS47] Aus diesem Grund ist ein Wechsel des Datenbanksystems nicht ohne hohen Aufwand durchführbar.

Seit der Version 4.1 unterstützt MySQL, sofern die Engine MyISAM verwendet wird, neben bekannten Datentypen wie Integer, Char, Float, Text oder Date auch die Speicherung von Geodaten. Hierzu wurden auf Spezifikationen des *OGC* basierende Erweiterungen in das Datenbanksystem implementiert. Mit diesen Erweiterungen wurde die Speicherung von geometrischen Objekten, wie Punkten (Point), Linien (Curve/Linestring) oder *Polygonen* sowie deren Verknüpfung in Form von Sammlungen(GeometryCollection) in MySQL eingeführt. Da die Geometrien in einer, für Menschen nicht lesbaren, Form abgespeichert werden, werden die von dem *OGC* festgelegten Formate Well Known Binary und Well Known Text ebenfalls unterstützt. Well Known Binary ist eine binäre Darstellung von Geometrien und ebenfalls nicht lesbar. Well Known Text bietet dagegen die Möglichkeit, Geometrien über Schlüsselworte zu beschreiben und somit lesbar darzustellen. Mit diesen Formaten können Daten in eine MySQL Datenbank eingefügt, bearbeitet

und abgefragt werden.[MYS11, Kar11]

Das *OGC* hat zur Arbeit mit Geodaten weitere Funktionen definiert, die eine *GIS*-kompatible Datenbank vorhalten muss. Obwohl es in MySQL möglich ist, Geodaten in Form von Geometrien zu speichern, werden diese Funktionen nicht oder nur zum Teil unterstützt. Es sind keine Koordinatentransformationen möglich, das bedeutet, dass die Sicherstellung eines konsistenten Formats nicht über das Datenbanksystem gegeben ist. Neben den Transformationen fehlen Funktionen zur Erzeugung neuer Geometrien aus bereits vorhandenen Geometrien. Für die Abfrage von Flächenbeziehungen gibt es zwar Funktionen jedoch arbeiten sie mit dem *Minimum Bounding Rectangle* (*MBR*), also mit dem kleinsten, umschließenden Rechteck. Je komplexer die Form einer Geometrie ist, desto ungenauer wird das Ergebnis einer Abfrage, zur Verdeutlichung ist in Abbildung 3.3 ein Beispiel zu sehen, das die Problematik anhand der Frage ob Punkte innerhalb Deutschlands liegen, darstellt. Laut einer Veröffentlichung auf MySQLForge, soll es aber erste Entwicklungsansätze für genauere Funktionen geben.[MYS11, Kar11, MYS23]

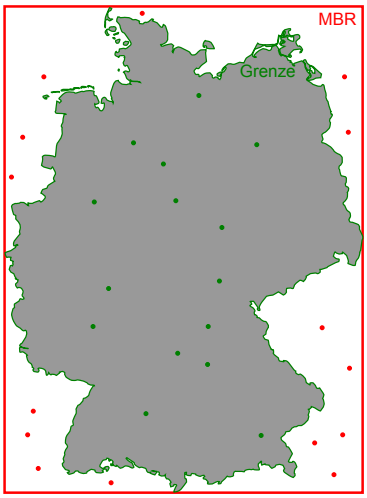

Abbildung 3.3: Ungenauigkeit des Minimum Bounding Rectangles
Quelle der BRD-Grenze: Portal der statistischen Ämter des Bundes und der Länder (DeStatis); David Liuzzo. [CC-BY-SA-2.0-de (www.creativecommons.org/licenses/by-sa/2.0/de/deed.en)], via Wikimedia Commons

Rechtlich gesehen wird MySQL in verschiedenen Versionen angeboten, die Version mit der Bezeichnung „Community“ ist Open Source und unter einer General Public Licence freigegeben.[MYS08]

Seit Oracle das Unternehmen Sun übernommen hat, ist der Fortbestand von MySQL allerdings fraglich, da Oracles Datenbanksystem in direkter Konkurrenz zu kostenpflichtigen Versionen von MySQL steht. Aufgrund der Befürchtung das

MySQL durch die Übernahme eingestellt wird, wurde unter anderem mit Hilfe einer Petition, versucht die Übernahme zu verhindern.[SMY24]

3.2.2 PostgreSQL

PostgreSQL ist, wie MySQL, ein relationales Datenbanksystem, es sind daher Gemeinsamkeiten zwischen PostreSQL und MySQL vorhanden. Da beide Systeme einen selben Zweck erfüllen und zudem zumindest in Teilen den SQL-Standards folgen, soll hier nicht noch einmal auf die Standardfunktionen eingegangen werden, stattdessen sollen die Abweichungen zwischen den beiden Systemen verdeutlicht werden.[BB17, MYS08]

PostgreSQL wird als leistungsstärkste Open Source Datenbank bezeichnet. [TAWAE08] Es unterstützt, wie MySQL, Abfragen zur Selektion und Aggregation von Daten in einer Datenbank. Auch in PostgreSQL können Nutzerrechte verwaltet und Views zur Einschränkung der Sicht auf eine Datenbank verwendet werden. Ein erster deutlicher Unterschied ist im Aufbau des Systems zu finden, im Vergleich zu MySQL bietet PostgreSQL keine Engines für das Datenbankmanagement, sondern nur ein fest implementiertes Datenbankmanagement. Dieses kann in seinem Leistungsumfang mit der InnoDB-Engine von MySQL verglichen werden, da es ebenfalls sichere Transaktionen und referentielle Integrität gewährleistet.[BB17, MYS08]

Generell sind PostgreSQL und MySQL zueinander inkompatibel, da MySQL wie erwähnt, zum Teil stark von den SQL-Standards zugunsten der *Performanz* abweicht, während PostgreSQL sehr genau an den Standards orientiert ist und somit kaum extra Funktionen mitbringt. Beide Systeme sind um sogenannte „Stored Procedures“ also vorgefertigte Funktionen erweiterbar, während MySQL einige optimierte Funktionen grundsätzlich mitbringt, können Funktionen in PostgreSQL nur durch eigene Erweiterungen erzeugt werden.[BB17, MYS08]

Auch PostgreSQL unterstützt verschiedene Programmiersprachen mit Treibern, wie einen JDBC-Treiber für Java und ist daher vielseitig einsetzbar.[BB17] Ein Vorteil von PostgreSQL ist die Unterstützung von Restores, mit dessen Hilfe eine PostgreSQL-Datenbank durch Log-Files wieder hergestellt werden kann. Dies ist allerdings für eine Webanwendung weniger wichtig, da in den meisten Fällen der direkte Zugriff auf das Datenbankmanagementsystem nicht erlaubt ist.[BB17]

Mit der Erweiterung PostGIS, entwickelt von Refractions Research[9], kann PostgreSQL zu einer räumlichen Datenbank erweitert werden. PostGIS erfüllt dabei die durch das *OGC* definierte „*Simple Features Specification for SQL*“[10]. Post-

[9] http://www.refractions.net/
[10] http://www.opengis.org/docs/99-049.pdf

greSQL unterstützt damit die in MySQL vorhandenen Datenformate für Geometrien sowie die Formate WKB und WKT für die Arbeit mit den Geometrien. Im Vergleich zu MySQL unterstützt PostGIS zusätzlich Funktionen zur Bildung neuer Geometrien, wie die Vereinigungsmenge und Schnittmenge zwischen Geometrien.[TAWAE08, PGS35] Die Abfragen von Flächenbeziehungen sind mit PostGIS ebenfalls genauer möglich, da die Berechnungen nicht auf Basis des *Minimum Bounding Rectangle* durchgeführt werden sondern anhand der Grenzen der verwendeten Geometrien.[PGS41]

Die Arbeit mit verschiedenen *Koordinatenreferenzsystemen* wird mit PostGIS durch die Tabelle „spatial_ref_system“ erleichtert. Sie ist Teil einer jeden räumlichen Datenbank in PostgreSQL und beinhaltet Parameter sowie dazugehörige EPSG-Codes zur Identifizierung und Umwandlung verschiedener *Koordinatenreferenzsysteme*. Jedes Koordinatensystem in dieser Tabelle wird mit Hilfe des EPSG-Code identifiziert. Die Anzahl beläuft sich auf ungefähr 3400 Einträge. Zusätzlich sind die Parameter gespeichert, die von der PROJ.4-Bibliothek zur Transformation in ein *Koordinatenreferenzsystem* benötigt werden. Mit Hilfe der *Projektions*bibliothek PROJ.4 können Koordinatentransformationen daher direkt mit PostGIS durchgeführt werden. Genaueres zur PROJ.4-Bibliothek ist unter dem Punkt Geoinformationen zu finden.[PGS41]

Um die volle PostGIS-Funktionalität zu nutzen, werden, wie bereits angedeutet, neben PostgreSQL weitere Werkzeuge, wie die erwähnte PROJ.4-Bibliothek verwendet. Die Geometriebibliothek GEOS[11], eine der Java Topology Suite[12] nachempfundene C++ Geometrie Engine zur Modellierung und Manipulation von zweidimensionalen Daten wird ebenfalls benötigt. Zum Import XML-basierter Geodaten, zum Beispiel *GML* ist zusätzlich die LibXML2-Bibliothek wichtig. LibXML2[13] ist eine auf der Programmiersprache C basierende Bibliothek die einen XML-Parser und Bearbeitungswerkzeuge für XML-Daten zur Verfügung stellt.[PGS41] Die eigentlichen Funktionen und Datentypen werden über PostgreSQL-Skripte in die PostgreSQL Datenbank eingefügt.[TAWAE08]

Mit der Erweiterung PostGIS wird PostgreSQL zu einer weitreichenden Lösung für die Arbeit mit Geodaten. Durch die *OGC*-konforme Umsetzung sind alle nötigen Funktionen vorhanden, um mit Geodaten arbeiten zu können. Unterstützt wird diese Aussage zusätzlich von den in PostGIS mitgelieferten Werkzeugen zum Import von Geodaten, wie zum Beispiel Shape-Dateien. Hinzu kommt, dass die meisten *GIS*-Werkzeuge die Anbindung an eine PostgreSQL Datenbank unterstützen.[TAWAE08]

[11] http://trac.osgeo.org/geos/
[12] http://sourceforge.net/projects/jts-topo-suite/
[13] http://xmlsoft.org/downloads.html

Ein Problem, dass sich bei PostgreSQL in Verbindung mit PostGIS ergibt, ist die erfahrungsgemäß geringe Verbreitung bei den Hosting-Anbietern. Häufig sind die mit PostgreSQL und PostGIS ausgestatteten Angebote im Vergleich zum typischen *LAMP*-Angebot teurer, da häufig nur ein „*root*"-Zugang zu einem Server die Möglichkeit bietet, alle nötigen Werkzeuge verfügbar zu machen. Obwohl PostgreSQL unter einer BSD-artigen Lizenz und PostGIS unter GNU-GPL steht, somit beide Teile Open Source sind, ist die Verbreitung relativ gering.[BB17, PGS35]

3.3 Geoinformationen

3.3.1 PROJ.4

PROJ.4 wird als eine der wichtigsten Bibliotheken im *GIS*-Bereich angesehen. Es ist eine in C geschriebene Bibliothek zur Umrechnung von Koordinaten in verschiedene Koordinatensysteme und *Projektionen*. Dazu besitzt PROJ.4 intern eine Liste vordefinierter Parameter der bekannteren Formate. Zusätzlich gibt es eine externe Liste mit EPSG-Codes und den nötigen Parametern zur Umrechnung. Es können auch eigene Parameter für weitere Formate verwendet werden. [TAWAE08] Parameter sind unter anderem in der bereits angesprochenen PostGIS-Tabelle „spatial_ref_sys" zu finden. Im Internet sind diese Parameter ebenfalls über die Spatial Reference[14] Seite zugänglich.

Zur Umwandlung von Koordinaten gibt es verschiedene Ansätze. Der Einfachste ist die Verschiebung des Mittelpunktes eines Koordinatensystems zum Mittelpunkt eines Anderen. Diese Methode ist sehr ungenau, da die Achsen des Ursprungssystems, nicht korrekt auf das Zielsystem transformiert werden, Abbildung 3.4 verdeutlicht dies anhand einer Verschiebung auf ein zusätzlich um eine Achse geneigtes Koordinatensystem.[TAWAE08]

Eine weitere Möglichkeit ist die Angabe mehrerer Parameter, die Verschiebung, Drehung und Anpassung des Maßstabs zum Zielkoordinatensystem beschreiben, diese Methode ist genauer. Da die Erde ein *Ellipsoid* ist, gibt es dennoch lokale Abweichungen. Die dritte Methode stellt daher eine Möglichkeit dar, die Umrechnungen lokaler Abweichungen ebenfalls vornehmen zu können. Dies wird ermöglicht, in dem eine Datei eingebunden wird, die die Differenzen zwischen den beiden Koordinatensystemen, für den zu transformierenden Punkt beschreibt. Diese Methode ist die genaueste Variante. PROJ.4 kann jede dieser Möglichkeiten ausführen. Zusätzlich kann mit PROJ.4 die Umwandlung der Koordinaten in eine *Projektion*, wie die Universale Transversale Mercator(UTM)-*Projektion*, durchgeführt werden. PROJ.4 steht unter der MIT-Lizenz, die der BSD-Lizenz ähnlich ist und eine freie Nutzung und Verbreitung der Software erlaubt, damit ist PROJ.4 Open Source.[TAWAE08]

[14] http://spatialreference.org/

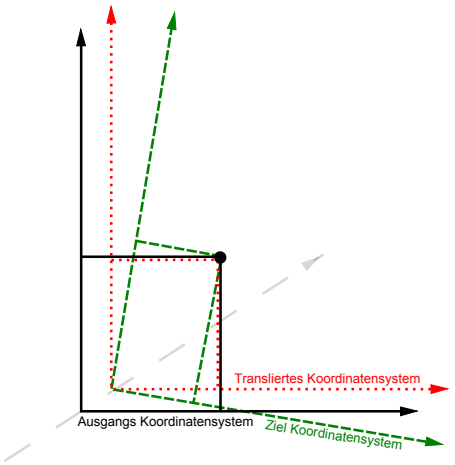

Abbildung 3.4: Ungenenaue Transformation durch eine Verschiebung des Mittelpunktes

Proj.4js

Um nicht nach einem Webserver mit installierter PROJ.4-Bibliothek suchen zu müssen und dennoch diverse Geodaten einbinden zu können, gibt es eine Umsetzung der PROJ.4-Bibliothek in JavaScript mit dem Namen Proj4js. Ebenso wie die originale PROJ.4-Bibliothek ist durch Angabe von Parametern eine Transformation von Koordinaten in diesem Fall Client-seitig möglich.[Ada14]

3.3.2 Kartenquellen

In der derzeitigen Umsetzung werden Karten, die aus Hintergrundkarten und *Vektordaten* bestehen, in der Webanwendung angezeigt. Die Hintergrundkarten sind als *Rastergrafiken* auf dem Webserver abgelegt. Um verschiedene Zoomstufen darstellen zu können, müssen die Rastergrafiken in verschiedenen Auflösungen vorliegen. Je höher die Auflösung ist, desto mehr Ressourcen werden durch eine Karte belegt. Da ein Webserver aber nur eine begrenzte Menge an Speicherplatz besitzt, ist dies ein Problem. Derzeit sind daher keine Zoomstufen in der Karte umgesetzt.

Eine Entlastung des Webservers bieten Karten von frei zugänglichen *Webservices*. Dafür kommen beliebige *Web Map Services* im Sinne des *OGC*, aber auch andere Services in Frage. Eine gute Alternative stellen Services da, die mittels einer API gekachelte Kartenbilder zur Verfügung stellen. Diese Services, im folgenden *Tile-Services* genannt, besitzen allerdings nicht immer die Genauigkeit, die von einer Karte erwartet wird. Genaue Karten hingegen gibt es vor allem in den Vermessungsämtern eines Landes. In den meisten Fällen werden diese Karten aber nicht in Form eines *Tile-Services* angeboten, dass Problem, der begrenzten Ressourcen, besteht bei deren Verwendung also weiterhin. Zusätzlich wird die Vermessung in einem Land meist dezentral organisiert, was dazu führt, dass die

Daten einzelner Bundesländer, in ihrer Qualität, durchaus voneinander abweichen können.[TAWAE08]

Die bekanntesten *Tile-Services* sind Google Maps, *Open Street Map*(*OSM*), Yahoo! Maps und Bing Maps. Damit Daten aus dem Internet genutzt werden können, müssen jedoch die damit verbundenen Richtlinien beachtet werden. Tatsächlich sind die mit den *Tile-Services* verbundenen Lizenzen nicht immer eindeutig. *Open Street Map* fällt nicht in diese Problematik, da es nicht nur frei zugänglich, sondern frei verfügbar im Sinne von Open Source ist. Zum Vergleich sollen die Eigenheiten der proprietären Dienste erläutert und im weiteren Verlauf mit *Open Street Map* in Beziehung gesetzt werden.[Mar10]

Google Maps

Google Maps ist der von Google betriebene Kartendienst. Google bietet verschiedene APIs an, um diesen Dienst auf der eigenen Webseite einzubinden. Für diese Arbeit ist nur die JavaScript-API interessant. Um die Google Maps API nutzen zu können, ist ein Nutzerkonto bei Google nötig und ein auf dieses Konto ausgestellter API-Schlüssel.[Mar10]

Mithilfe der API ist es möglich statische oder dynamische Karten mit Bedienelementen, anzufordern und diese mit Symbolen zu versehen. Die Umwandlung von Orten in Koordinaten ist ebenfalls über die Google Geocoding API möglich. Bei der Anzeige der Daten gibt es die Möglichkeit zwischen der Darstellung „Satellit“, wahlweise mit oder ohne Daten über Straßen und Orte, und „Karte“ zu wählen. Die Karten werden in vorgerenderten Kacheln dargestellt. Es sind Zoomstufen von $0 - 21+$ auswählbar, bei $21+$ stellt eine doppelte Längeneinheit etwa 20 Meter dar. [GAP19]

Google bietet die Daten kostenlos für nicht gewerbliche Zwecke an, jedoch gibt es bei dieser Lizenz Einschränkungen. Zum einen ist es Google erlaubt, in den Karten Werbung zu veröffentlichen, zum anderen beschränkt Google zumindest die Nutzung des Geocoding Dienstes auf 15.000 Anfragen pro Tag.[Mar10]

Yahoo Maps

Dies ist der Kartendienst von Yahoo!. In den Darstellungsmöglichkeiten unterscheidet er sich nicht besonders von Google. Es kann zwischen „Karte“ und „Satellit“ gewählt werden, die Option „Satellit“ mit zusätzlichen Kartendaten, hat einen eigenen Namen und wird mit „Hybrid“ ausgewählt. Die Maximale Zoomstufe ist 12 in der eine Längeneinheit 50 Metern entspricht.[YAP42]

Auch Yahoo! bietet einen Zugang zu den Karten in Form eines *Webservice*. Zur Nutzung dieses *Webservice* ist genau wie bei Google ein Yahoo! Nutzerkonto

und ein Schlüssel, von Yahoo! als „Application ID“ bezeichnet, nötig. Auch Yahoo! bietet verschiedene APIs an, dazu zählt unter anderem eine für diese Arbeit interessante *AJAX*-API. Ein Geocoding-Dienst in Form von Yahoo! PlaceFinder ist ebenfalls vorhanden, die Anfragenmenge ist ebenfalls begrenzt, allerdings auf 50.000 Anfragen pro Tag.[YAP42]

Die Nutzung der Daten ist bei Yahoo! kostenlos und darf nur im privaten Sinne geschehen, eine gewerbliche Nutzung ist also ausgeschlossen. [Mar10]

Bing Maps

Der von Microsoft betriebene Kartendienst Bing Maps unterscheidet sich nicht von den anderen proprietären Diensten, es kann ebenfalls zwischen einer Karte als Menüpunkt „Straße“ und einer „Vogelperspektive“, bei den anderen beiden Diensten als „Satellit“ bezeichnet, gewählt werden. Eine Hybridansicht aus Karte und Satellit ist ebenfalls möglich. Die Zoomstufen gehen vom Meeresspiegel bis zu 20.000.000 Metern über dem Meeresspiegel.[BAP05]

Auch Microsoft bietet für den Zugriff auf die Bing Maps Daten eine JavaScript-API an, die in Verbindung mit einem Windows Live Konto und einem Schlüssel nutzbar ist.[BAP05]

Das Lizenzmodell von Bing Maps ist, im Vergleich zu den anderen Diensten, sehr fein, es wird zwischen verschiedenen Nutzungsmodellen unterschieden. Für eine rechtliche Absicherung wird von den Autoren des Buches OpenLayers, Marc Jansen und Till Adams empfohlen vorher Rat bei einem Juristen zu suchen.[Mar10]

Open Street Map

Open Street Map(*OSM*) ist, im Gegensatz zu den zuvor beschriebenen proprietären Diensten, ein freier Dienst im Sinne von Open Source. *OSM* steht unter der Creative Commons Attribution-ShareAlike Lizenz und ist ein Projekt, das ähnlich Wikipedia[15] zum Ziel hat, Informationen in diesem Fall Geodaten für jeden zugänglich zu machen. Dabei verfolgt es denselben, als „Crowd-Sourced“ bezeichneten Ansatz wie die genannte Enzyklopädie, in dem es jeden Menschen dazu aufruft, Geodaten zu sammeln und mitzuteilen. Dies wird dadurch ermöglicht, dass die Daten von *OSM* keinem speziellen Vermessungsverfahren unterliegen, sie werden stattdessen mit Hilfe von GPS-Geräten zusammengetragen. Neben GPS-Daten kann auch jede als frei im Sinne von Open Source charakterisierte Quelle herangezogen werden. Luftbilder oder freigegebene Karten wären zum Beispiel ebenfalls mögliche Quellen. Neben dem Angebot alle in *OSM* aufgenommenen

[15] http://de.wikipedia.org/wiki/Wikipedia:Hauptseite

Daten herunterzuladen, bietet *OSM* für Webanwendungen auch Karten in gekachelter Form an. Zur Abfrage und Darstellung kann zum Beispiel OpenLayers eingesetzt werden. Der Bezug von Kacheln ist aber auch mit der Google Maps API möglich. Im Gegensatz zu den proprietären Diensten ist es bei *OSM* nicht möglich Satellitenbilder anzeigen zulassen, alle Daten von *OSM* sind *Vektor-basierte Daten*. [RT10, Mar10]
Für die Bereitstellung der Kacheln(Tiles) gibt es zwei Servertypen.

Mapnik ist einer dieser Server. Mapnik bietet gerenderte Tiles in einer begrenzten Anzahl von Zoomstufen an, zusätzlich filtert Mapnik die darzustellenden Details abhängig von der Zoomstufe, um die Karten übersichtlich zu halten. Der Server basiert auf einer PostGIS Datenbank, die regelmäßig mit den aktuellsten *OSM*-Daten aktualisiert wird. Über eine URL[16] kann mit dem Server kommuniziert werden, um Daten anzufordern. Ein Vorteil ist die Möglichkeit des Servers spezielle Zoomstufen, die nicht vorgehalten sind, auf Anfrage zu rendern. Eine Besonderheit besteht darin, dass dem Server mitgeteilt werden kann, dass eine Kachel neu gerendert werden soll.

Um die *Performanz* zu verbessern, können verschiedene Tiles über Subdomains geladen werden, durch diese simultanen Anfragen kann die Ladezeit einer Webanwendung verkürzt werden.[RT10]

Der zweite Typ ist Osmarender, dieser Server wird in dem *OSM* verwandten Projekt Tiles@Home eingesetzt, bei dem das Rendern der Kacheln dezentral ausgeführt wird, während die Kacheln zentral abgerufen werden können. Auch Osmarender, kann über die URL[17] auf verschiedenen Subdomains angesprochen werden, um Kacheln anzufordern. Im Vergleich zu Mapnik bietet Osmarender keinen angepassten Detailgrad je Zoomstufe, da im Fall von Osmarender *Vektordaten* vorliegen. Die Unterteilung der Zoomstufen unterscheidet sich ebenfalls. Während Osmarender mehr Zoomstufen als Mapnik vorhält, ist die höchste Zoomstufe von Mapnik mit 20 Metern pro Längeneinheit deutlich stärker als die von Osmarender mit 50 Metern pro Längeneinheit.[RT10, OSM38]

Neben den Servern für Karten bietet *OSM* mit Nominatim, wie auch Yahoo! Oder Google , einen Dienst zur Suche von eingetragenen Orten an. Die API ist ebenfalls über eine URL[18] ansprechbar.[RT10] Die Abfragemenge beschränkt sich bei Nominatim auf die Bitte, nicht mehr als eine Abfrage pro Sekunde zu tätigen.[OSM38]

Ein Nachteil und gleichzeitig ein Vorteil von *OSM* gegenüber den proprietären Diensten ist die Quelle der Daten. Aufgrund der Offenheit des Projektes und

[16] http://[Subdomain.]tile.openstreetmap.org/Zoomstufe/X/Y.png
[17] http://[Subdomain.]tah.openstreetmap.org/Tiles/tile/Zoomstufe/X/Y.png
[18] Http://http://nominatim.openstreetmap.org/search

der abweichend genauen GPS-Bestimmungsmöglichkeiten wird häufig über die Integrität der Daten diskutiert. Neben dieser Diskussion ermöglicht diese Art der Datenerhebung aber jeder Person, fehlende Daten auf einfache Weise selbst zu ergänzen. Dieser Vorteil ist besonders in Ländern ohne ein gut funktionierendes Vermessungswesen von Gewicht, da private Personen stattdessen die Möglichkeit haben in *OSM* für eine Erfassung zu sorgen.[RT10] Ein Projekt zur Förderung dieser Arbeit ist das durch *OSM* gestartet GPStogo[19].

Durch die Lizenz ist die Nutzung und Veränderung der Karten erlaubt, solange *OSM* als Quelle angegeben wird. Eine Registrierung bei *OSM* ist im Gegensatz zu Google und anderen Anbietern nur nötig, wenn man auch Daten einbringen möchte, für die Abfrage der Kacheln, muss keine Registrierung erfolgen.[RT10]

Mapserver

Da im Projektteam zukünftige Überlegungen über Funktionen, die nur mit einem *Mapserver* realisiert werden können existieren, soll das Thema kurz dargestellt werden. Ein *Mapserver* wird aber durch das Projektteam, im Rahmen dieser Arbeit, aus Kostengründen vorerst nicht gewünscht. Im Rahmen der Lösungskonzepte für eine Umsetzung ist ebenfalls eine Konzeptidee zu diesem Thema vorhanden.

Es gibt derzeit zwei *Mapserver* im Open Source Bereich die weitgehend bekannt sind, der UMN MapServer und der auf Java basierende Geoserver. Während der Geoserver eine auf Java-Servlets basierende Umsetzung ist,[Pu43] ist der UNM MapServer eine Anwendung die auf das Common Gateway Interface(CGI) aufbaut.[MFB46]

Ein *Mapserver* bietet durch die *OGC* spezialisierte *Webservices*, sogenannte *OpenGIS Web Services*, an. Die *OGC* definiert drei Servicearten. Der *Web Map Service*(*WMS*) wird zur Anforderung und Auslieferung von dynamisch erstellten Karten definiert, über ihn können gerenderte Kartenansichten bezogen werden. Der *Web Feature Service*(*WFS*) kann Vektor-basierte Daten (Features) liefern, dazu kommuniziert er in einem XML-Format, meist *GML*, und stellt XML-strukturierte Daten zur Verfügung. Die Art der Darstellung ist in diesem Fall vom Client abhängig. Ein *Web Coverage Service*(*WCS*) ist definiert, um XML-basierte *Rasterdaten* zur Verfügung zu stellen, dieser Service ist somit das Gegenstück zum *WFS*.[TAWAE08] Für den *WFS* ist zusätzlich eine erweiterte Form definiert. Während alle genannten Services nur Datenanfragen ausführen können aber keine Möglichkeit bieten neue Daten aufzunehmen, gibt es den *WFS* in der Variante *WFS-T*, wobei „T“ für „Transactional“ steht. Der *WFS-T* ermöglicht es, *Vektordaten* auf einem *Mapserver* zu verändern beziehungsweise anzulegen. Der UMN MapServer unterstützt jedoch keinen *WFS-T*.[TAWAE08] Mit Geoserver kann ein

[19] http://wiki.openstreetmap.org/wiki/GPStogo

WFS-T genutzt werden, um den Geoserver zu einem dynamischen *Mapserver* und Geodatenspeicher zu machen.[TAWAE08]

3.4 Folgerungen

Die Betrachtung von Joomla! und dessen Komponenten hat ergeben, dass es möglich ist, *GIS*-Anwendungen mit Joomla! umzusetzen. Dieses Ergebnis wird besonders von der erwähnten Erweiterung WISroGIS unterstützt. Joomla!, sofern möglich, als *WCMS* zu verwenden ist auch Teil im Sinne der Nutzer. Dies begründet sich durch die einfache Handhabung von Joomla! und den anfallenden Portierungsaufwand bei einem Wechsel zu einem anderen System, da Joomla! in der derzeitigen Umsetzung eingesetzt wird. Aus diesem Grund wurde auf eine eingehende Betrachtung weiterer *WCMS* verzichtet.

In der Charakterisierung von MySQL wurde jedoch deutlich, dass das einzige von Joomla! unterstützte Datenbanksystem, nur eingeschränkte Möglichkeiten besitzt, um mit Geodaten zu arbeiten. Ein weiteres Problem sind die Spekulationen, die aufgrund der Übernahme von Sun durch Oracle um die Zukunft von MySQL entstanden sind. Diese Probleme haben dazu geführt, zumindest noch ein weiteres Datenbanksystem, neben MySQL, zu betrachten, obwohl Joomla! originär nur MySQL unterstützt. Bei der Betrachtung von PostgreSQL wurde deutlich, dass PostgreSQL und MySQL für den Anwendungsbereich dieser Arbeit, in ihren Grundfunktionen kaum verschieden sind, solange für MySQL die Engine InnoDB genutzt wird. Beide Datenbanksysteme unterstützen die nötige Funktionalität für eine Webanwendung. Tatsächlich ist aber im Umgang mit Geodaten nur die Engine MyISAM verwendbar, was zu einer Einschränkung im Bereich der Datenbankintegrität führen kann.

Im Umgang mit Geodaten ist PostgreSQL mit der Erweiterung PostGIS jedoch weitaus umfangreicher als MySQL. Als besonderer Vorteil, wird die integrierte Transformationsfunktion erachtet, sie ermöglicht ohne weitere Eingriffe eine einheitliche Speicherung von Geodaten. Da für den vollen Funktionsumfang der Erweiterung PostGIS jedoch weitere Werkzeuge auf dem Server vorhanden sein müssen, ist eine Standardinstallation von PostgreSQL nicht ausreichend. Als Folge wird ein Webserver benötigt, der weitaus teurer ist, als ein Webserver mit einer typischen *LAMP*-Konfiguration. Da MySQL ohne Erweiterungen auskommt, ist in dem Fall eine *LAMP*-Konfiguration ausreichend und die Verwendung einer MySQL Datenbank kostengünstiger. Dafür ist mehr Programmierarbeit notwendig um fehlende Funktionen dennoch anbieten zu können und die Datenbankintegrität zu gewährleisten.

Da die Betrachtung von *Mapservern* aufgrund zukünftiger Ideen ebenfalls vorgenommen wurde, stellte sich heraus, dass auf eine lokale Datenbank ganz verzichtet

werden könnte, wenn ein *WFS-T*, wie er mit Geoserver möglich ist, eingesetzt werden würde.

Es ist nicht abwegig im Hinblick auf zukünftige Fähigkeiten der Anwendung einen solchen Ansatz der Datenablage zu wählen, da zukünftig auch *GIS*-Funktionen gewünscht werden könnten, die nicht mit einfachen Mitteln zu bewerkstelligen sind. Die Verwendung von Geoserver würde es ermöglichen, Geodaten Client-unabhängig bereitzustellen. Neben einer Webdarstellung wären auch Smartphone Apps denkbar, die durch einen *Mapserver* auf denselben Datenbestand zugreifen könnten. Die Abfrage von Daten sowie die direkte Eintragung von Fundorten über ein mit GPS ausgestattetes, internetfähiges PDA, wäre ein Anwendungsszenario. Eine genauere Diskussion dieser Möglichkeiten ist Bestandteil der Konzepte.

Für die Darstellung einer dynamischen Karte mit Orten, sind die als Geodaten gespeicherten Orte alleine nicht ausreichend, es ist eine Karte nötig, die den örtlichen Bezug verdeutlicht. Aus diesem Grund wurden verschiedene Kartenquellen untersucht. Statische Karten, wie sie derzeitig in dem Atlas verwendet werden, fallen aufgrund der Forderung nach einer Karte mit verschiedenen Zoomstufen weg. Um den Webserver zu entlasten, wurden *Webservices* zur Lieferung von Karten in gekachelter Form betrachtet. Auch wenn diese Karten nicht zwingend so genau sind wie Karten einer Vermessungsanstalt, bieten sie dennoch eine flächendeckend einheitliche Kartendarstellung, was bei Karten von Behörden nicht der Fall sein muss, da die Vermessung zum Beispiel in Deutschland nicht auf Bundesebene stattfindet. Dieses Thema wird umso komplizierter, wenn man die Darstellung mehrerer Länder betrachtet. Daher ist es durchaus sinnvoll, einen im Internet verfügbaren *Webservice* zu nutzen.

Bei der Untersuchung stellte sich heraus, dass bei den proprietären Services in jedem Fall eine Registrierung erforderlich ist, um überhaupt Karten anfordern zu dürfen. Hinzu kamen, zum Teil komplexe Lizenzen und die Möglichkeit in Zukunft Werbung in einer eingebundenen Karte akzeptieren zu müssen. Dagegen existiert das *Open Street Map* Projekt, durch das Karten frei zugänglich sind. Kartendaten dieses Projektes wird jedoch des öfteren, aufgrund der Art ihrer Erfassung, Ungenauigkeit vorgeworfen. Dagegen steht allerdings der Vorteil, falls nötige Geodaten nicht oder ungenau vorhanden sind, eigene Daten in diesem Projekt erfassen zu können. Bei den proprietären Diensten besteht in diesem Fall eine Abhängigkeit vom Betreiber. Neben den objektiven Betrachtungen muss eine subjektive Betrachtung einzelner Dienste bedacht werden. Ein Datenlieferant mit Namen Google, stellt möglicherweise in einer wissenschaftlichen Community ein größeres Konfliktpotential dar, als ein Ansatz auf Basis von *Open Street Map*.

Die gewonnenen Kenntnisse zeigen vor allem ein Problem sehr deutlich, Joomla! ist mit der derzeitigen Version nur in Verbindung mit MySQL nutzbar. Die Frage nach einem Wechsel zu einem anderen *WCMS/CMS* zur besseren Unterstützung

von Geodaten wurde jedoch schon in der vorhergehenden Analyse verneint. Dies hat vor allem den Grund, dass mit einem neuen *CMS*, sei es auch ein GeoCMS das explizit für den Umgang mit Geodaten konzipiert ist, neue Problemstellungen auftreten. Zum einen müssen Daten die in der Webseite neben den Geodaten existieren portiert und zum anderen müssen die Teammitglieder in ein neues System eingearbeitet werden. Zusätzlich gibt es auch in den anderen Systemen Schwächen. Ein GeoCMS zum Beispiel legt großen Wert auf das Management von Kartenebenen. Da im Rahmen des Projektes neben Geodaten jedoch vor allem Informationen zu den biologischen Arten und dem Projekt an sich vermittelt werden sollen, müssen neben den Geodaten diese Daten verwaltet werden. Ein GeoCMS sieht die Funktionen dafür nicht vor. Durch die Verwendung eines normalen (W)*CMS* lassen sich Geodaten zwar nicht ohne Weiteres verwalten, dafür werden aber die Probleme in Verbindung mit der Verwaltung von Texten und verschiedenen Medientypen wie Bildern und Dokumenten durch das *CMS* gelöst. Für hybride Webanwendungen wären damit mehrere Probleme abgedeckt, während ein GeoCMS nur die Probleme im Umgang mit Geodaten abdeckt. Bei den generellen (W)*CMS* neben Joomla! zählen andere Probleme, wie das Fehlen eines Backends oder die Spezialisierung auf gewisse Darstellungsformen, wie einem Blog oder ein Wiki. Obwohl Joomla! mit MySQL nicht die optimale Unterstützung von Geodaten bietet, ist es in den anderen Bereich von Vorteil dieses System zu verwenden.

Bei näherer Betrachtung der derzeitigen Anforderungen für die Anwendung stellte sich außerdem heraus, dass die nötigen Funktionen mit Hilfe der vorgestellten Werkzeuge OpenLayers und Proj4js auch anderweitig umgesetzt werden können. Mit Hilfe von Proj4js, ist es sogar möglich den Server zu entlasten, in dem die Transformationen im Client durchgeführt werden. Gleichzeitig ist dadurch, die angesprochene Einheitlichkeit der Geodaten gewährleistet. Mit Hilfe von OpenLayers kann Joomla! um die flexible Darstellung von Geodaten erweitert werden, ohne das in Joomla! tiefgreifende Veränderungen nötig sind, die spätere Updates auf neue Versionen unmöglich machen. Dieses Problem wäre bei dem Versuch Joomla! auf PostgreSQL umzustellen aufgetaucht, da alle Datenbankanfragen des Systems verändert werden müssten.

Da die Entwicklung von Joomla! Tendenzen zur Datenbanksystemunabhängigkeit aufweist und in der Version 1.6 Beta bereits erste Versuche zu einer Abstraktionsschicht[20] unternommen werden, soll die zukünftige Anwendung so implementiert werden, dass später ein Wechsel zu PostgreSQL oder sogar einem *Mapserver* möglich bleibt.

[20] http://joomlacode.org/gf/project/joomla/scmsvn/?action=browse&path=/development/branches/database/

3.5 Lösungsansätze

Aus den Erkenntnissen der vorhergehenden Analyse ergeben sich verschiedene Lösungsansätze. Während ein Lösungsansatz durch Verwendung eines anderen *CMS* bereits ausgeschlossen wurde und Joomla! nicht, ohne die Möglichkeit zu verlieren Versionsupdates durchzuführen, mit PosgreSQL als primären Datenspeicher verbunden werden kann, sind drei Lösungsansätze denkbar.

- Erweiterung der *GIS*-Funktion von MySQL durch die Webanwendung
- Parallele Datenbanken
- Auslagerung der Geodaten auf einen *Mapserver*

Die Einführung paralleler Datenbanken, MySQL für die Daten von Joomla! und PosgreSQL für die Geodaten auf einem Webserver, ist von diesen Lösungsansätzen, die am wenigsten sinnvolle. Da kein typischer Webserver zwei Datenbanksysteme vorhält, müsste ein eigener Root-Server oder eine externe Datenbank verwendet werden. Ein weiteres Problem liegt darin, dass in der Joomla!-Konfiguration keine zweite Datenbank vorgesehen ist, daher müsste die Webanwendung die Verbindungsdaten selbst verwalten. Zusätzlich gewinnt die Anwendung unnötig an Komplexität, da neben der Verwaltung von zwei Datenbankensystemen, zum Teil Verbindungen zwischen diesen Datenbanksystemen nötig sind, um Joomla!-Strukturen mit Geodaten in Beziehung setzen zu können. Daraus resultiert, dass Abfragen, die Daten aus beiden Datenbanksystemen benötigen, langsamer sind, da zwei Verbindungsanfragen nötig wären. Auf Serverebene können parallele Datenbanksysteme zusätzlich dazu führen, dass sich diese in ihrer *Performanz* gegenseitig beeinträchtigen.

Ein im Grunde ähnlicher Ansatz ist die Auslagerung der Daten auf einen *Mapserver*. Obwohl die Daten ebenfalls in einer separaten Datenbank liegen, sind nicht dieselben Nachteile vorhanden. Tatsächlich ist die Lösung sogar in bestimmten Anwendungsszenarien von Vorteil. Abbildung 3.5 zeigt einen exemplarischen Aufbau eines Systems aus *Mapserver* und Webclient.

In diesem System werden alle Daten die mit den Geodaten in Verbindung stehen auf einen *Mapserver* vom Typ Geoserver mit einer PostGIS-Datenbank ausgelagert. Der Geoserver sorgt durch ein *WMS* dafür, dass die ausgelagerten Daten in gerenderter Form angefordert werden können. Ein *WFS-T* ist dafür zuständig, die Zusatzinformationen der einzelnen Geodaten in Form von Features bereitzustellen. Der *WFS-T* kann zusätzlich dazu genutzt werden, Daten auf dem Geoserver zu bearbeiten oder neue Daten zu erzeugen. Obwohl der *WMS* auch zu Bereitstellung von Hintergrundkarten verwendet werden kann, ist dies nicht zwingend notwendig. Theoretisch ist es sogar sinnvoller die Hintergründe weiterhin wie dargestellt, in Form von Tiles in diesem Fall von *OSM* zu beziehen, da hierdurch

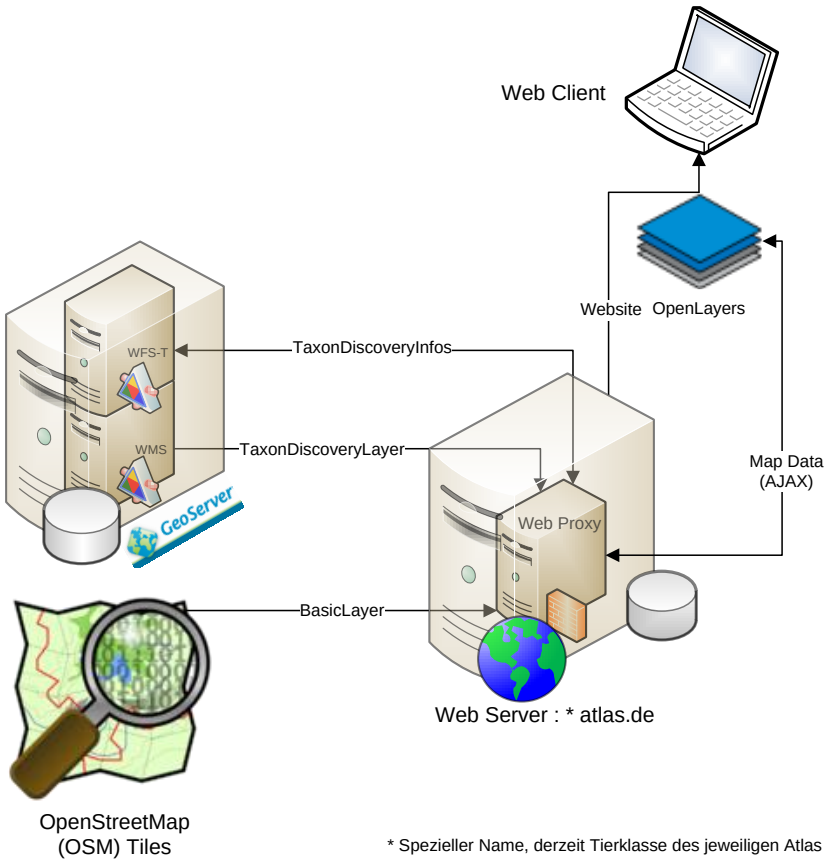

Abbildung 3.5: Architektur mit Webclient und GeoServer als Mapserver

die Last auf dem eigenen Geoserver verringert werden kann. Der Webserver realisiert in diesem Ansatz eine Vermittlungsstelle. Über ihn wird eine Verwaltung zur Erzeugung eines Webclients realisiert. Für den Einsatz von Joomla! bedeutet dies, dass über das Backend die Verbindung zu dem Geoserver verwaltet und ein Layermanagement ermöglicht wird, das die angebotenen Layer des Geoservers den Seiten im Frontend zuordnen kann. Das Frontend stellt den eigentlichen Client dar. Mithilfe der Integration von OpenLayers in das Joomla!-*CMS* werden die, im Backend zugewiesenen, Layer zusammengefügt und dargestellt. Da mit *AJAX* aufgrund der „*Same Origin Policy*“ von JavaScript nicht direkt mit Daten anderer Server gearbeitet werden kann, muss in Joomla! ein *Proxy* implementiert sein, der die Anfragen an den Geoserver weiterleitet. Neben den Funktionen für die Anfrage von Daten werden in Joomla! zusätzlich Funktionen für das Senden von Daten an den *WFS-T* implementiert, womit das Einfügen von Geodaten aus dem Webclient möglich wird. Durch den Geoserver als Schnittstelle für Geodaten wären andere Clientarten denkbar, da die Daten von dem Webserver unabhängig sind.

Ein Nachteil entsteht jedoch durch die Notwenigkeit der Eintragung oder Veränderung von Geodaten. Während für die Abfrage einer Karte, eingeschränkte Views einer Datenbank verwendet werden können, ist für Eintragungen über einen *WFS-T* immer eine komplette Tabelle inklusive Primärschlüssel nötig. Dies scheint eine derzeitige Einschränkung im Geoserver zu sein. Das bedeutet, zum Eintragen der Daten müssen komplette Tabellen der Datenbank über den Geoserver zugänglich gemacht werden. Damit wird durch den Geoserver implizit eine Art *Webservice* zur entfernten Verbindung mit einer Datenbank dargestellt, diesbezüglich auftauchende Sicherheitsfragen für Datenbanken und Server müssen vorher geprüft werden.

Für diese Art der Realisierung fehlt es jedoch derzeit an Mitteln, da mehrere Server nötig wären und durch die beschriebenen *Webservices*, einfache Hosting-Angebote nicht mehr ausreichend wären. Aus diesem Grund wird der Ansatz in dieser Arbeit nicht weiterverfolgt, er könnte aber ein zukünftiges Thema des Projektes darstellen.

Daraus ergibt sich, für diese Arbeit der Ansatz einer Webanwendung auf Basis von Joomla! und MySQL, wobei die *GIS*-Funktionen von MySQL bei Bedarf durch entsprechende externe Funktionen in Joomla! erweitert werden müssen. Wie bereits erwähnt, soll die Umsetzung jedoch so generisch gehalten werden, dass in Zukunft Szenarien, wie ein Wechsel des Datenbanksystems oder eine *Mapserver*-Architektur wie die Beschriebene mit geringerem Aufwand möglich sind.

4 Realisierungskonzept

Wie aus den Lösungskonzepten zu entnehmen ist, geht es darum einen digitalen Atlas zur Erfassung und Visualisierung von Biodiversitätsdaten mit einem möglichst generischen Ansatz zu implementieren. In diesem Atlas soll es berechtigten Nutzern möglich sein, Fundorte biologischer Arten einzutragen, diese Fundorte werden auf einer Karte visualisiert. Zusätzlich soll es die Möglichkeit geben Informationen und Bilder einer Art abzurufen. Die Implementierung soll so erfolgen, dass ein Joomla!-Grundsystem um die angesprochene Atlasfunktionalität erweitert werden kann.

Die Anforderungen in dieser Arbeit sind prinzipiell am *Userinterface* orientiert. Im *CMS* Joomla! wird das *Userinterface* in die Bereiche Backend und Frontend unterschieden. Das Frontend stellt, die öffentlich zugängliche Webseite dar, während das Backend zur Verwaltung der Daten verwendet wird, die für eine Darstellung im Frontend benötigt werden. Je nach Aufgabe werden die Anforderungen daher diesen Bereichen zugeordnet. Anforderungen gehören zum Bereich Backend, wenn sie zur Verwaltung der Daten beitragen und nur durch bestimmte Nutzer in Anspruch genommen werden sollen. Anforderungen des Frontends sind dagegen hauptsächlich auf die Darstellung bezogen und in den meisten Fällen öffentlich. Einige Anforderungen werden im Backend sowie im Frontend genutzt, da deren Verwendung allerdings voneinander abweicht, werden sie im Backend und im Frontend einzeln aufgeführt. Manche Anforderungen des *Userinterface* benötigen für die Umsetzung, weitere systeminterne Funktionen. Eine systeminterne Funktion wird diesbezüglich gesondert unter dem Punkt Plugin betrachtet. Um möglichst wenig an den Arbeitsgewohnheiten der bisherigen Nutzer zu ändern, wird versucht, alte Funktionen, sofern möglich und nötig, in die neue Komponente zu integrieren.

Zur Sicherstellung der Integrität aller Daten muss das in den Anforderungen beschrieben Nutzermodell berücksichtigt werden. Die Anforderung unterscheidet zwischen mindestens drei Nutzertypen, hier mit einem möglichen Bezug auf die Joomla!-Bereiche dargestellt:

- **Gast** – Anonymer Besucher der Website (Frontend)
- **Registriert** – Bekannter Nutzer mit eingeschränkten Rechten (Frontend)
- **Administrativ** – Nutzer mit allen Rechten (Backend & Frontend)

Aus der Analyse ging hervor, dass Joomla! ein vordefiniertes Nutzermodell, in Form von Gruppen, in die ein Nutzer eingeordnet werden kann, bietet. Dieses Modell sollte, sofern dies möglich ist, verwendet werden.

4.1 Frontend

Das Frontend stellt die eigentliche Webseite dar, die für jeden Internetnutzer erreichbar ist. Daher werden alle Forderungen, die öffentlich sowie für einen größeren bekannten Nutzerkreis zugänglich sind, dem Frontend zugeordnet. Die Darstellung im Frontend wird bei Joomla! durch auswählbare Ansichten ermöglicht. Die Ansichten sollten im Rahmen einer Komponente implementiert werden, da sie so über das Joomla!-Backend in das Menü des Frontends eingebunden werden können und zusätzlich leichter zu organisieren sind.

Zu den Ansichten gehört vor allem der Zugang und die Darstellung der Arten. Der Zugang zu einzelnen Arten kann am besten durch eine Liste von Links organisiert werden, die durch die Komponente automatisch erstellt wird und in der Artenansicht sichtbar ist. Für die in den Anforderungen angesprochenen Sprachen einer Artenauswahl ist ein Schaltmechanismus denkbar.

Da die derzeitige Umsetzung und auch die zukünftige Umsetzung einen Atlas darstellen soll, ist die Karte der Mittelpunkt der Artenansicht im Frontend. Die Karte muss aus diesem Grund einen möglichst großen Platz zur Verfügung bekommen. Der Stellenwert der Karte zeigt sich vor allem darin, dass vonseiten des Projektteams andere Informationen zugunsten der Karte ausgelagert werden dürfen. Aus den Anforderungen geht hervor, dass eine Karte für die Darstellung von Vorkommen einer Art aus verschiedenen Ebenen bestehen kann und eine vordefinierte Ansicht besitzt. Die Karte muss auch die Auswahl einzelner Ebenen zulassen. Zusätzlich müssen die in den Anforderungen angesprochenen Markierungen in der Karte umgesetzt werden.

Für die Bereitstellung der geforderten Funktionen können wie in der Analyse gezeigt verschiedene APIs von im Internet verfügbaren Kartendiensten verwendet werden. In der Analyse wurde allerdings gezeigt das mit der JavaScript-Bibliothek OpenLayers neben diesen Funktionen auch verschiedene Quellen für eine Karte verwendet werden können. Daher sollte diese Bibliothek verwendet werden. Mit Hilfe der Kontrollelemente, die durch diese Bibliothek zur Verfügung gestellt werden, sind typische Navigationsanforderungen genauso möglich, wie das Auswählen spezifischer Layer durch den Nutzer. Eine Zoomfunktion ist ebenfalls vorhanden. Um die durch den Nutzer gewählte Zoomstufe einer Karte gemäß den Forderungen auch nach dem Aufruf einer neuen Artseite in der Karte anzeigen zu können, sollte ein *Browser-Cookie* verwendet werden. Für die Darstellung der geforderten Markierungen können mit OpenLayers, verschiedene vektorbasierte Formen, mit

individuellen Farben genutzt werden.

Im Rahmen des geforderten Umweltschutzes muss in der Ansicht zusätzlich zwischen Gast und angemeldetem Nutzer unterschieden werden. Hierzu kann das durch Joomla! für jeden Benutzer angelegte Nutzerobjekt verwendet werden. Das Nutzerobjekt besitzt ein Attribut „guest“ wenn dieses wahr ist, ist der Nutzer dem System nicht bekannt. Ist ein Nutzer nicht bekannt, bekommt er, falls eine Verfremdung von Daten möglich ist, diese verfremdeten Daten in der Karte angezeigt. Auf die Verfremdung der Daten wird noch einmal genauer im Punkt Plugin eingegangen. Registrierte Nutzer können dadurch von Gästen unterschieden werden, so dass für sie genauere Daten verwendet und eigene Daten hervorgehoben werden können.

Um die geforderten Informationen bei dem Überfahren eines Punktes der Karte mit der Maus angezeigt zu bekommen, kann, falls OpenLayers verwendet wird, das Kontrollelement „SelectFeature“ genutzt werden, um einen Popup-Effekt zu erzeugen.

Alle weiteren Artinformationen sollten unabhängig von der Karte aus der Datenbank abgefragt und zur Darstellung mit Hilfe eines *Templates* aufbereitet werden.

Neben der Darstellung besteht die zweite Ansicht des Frontends aus der Möglichkeit „*User Generated Content*“ in den Atlas aufzunehmen. Dazu ist ein Formular für registrierte Nutzer nötig. Der Nutzer kann die biologische Art des Fundes über eine Liste aus wissenschaftlichen Namen in dem Formular auswählen. Um den Fundort des Vorkommens auf der Karte darstellen zu können, sind in dem Formular, Felder für die Koordinaten des Fundortes erforderlich.

Koordinaten müssen nicht zwingend aus demselben *Koordinatenreferenzsystem* stammen. Es wird aber ein einheitliches Format für eine konsistente Speicherung und Darstellung benötigt. *WGS84* als internationales *Koordinatenreferenzsystem* sollte dieses einheitliche Format sein. Da die Durchführung einer Koordinatentransformation aber nicht trivial ist, soll der Nutzer nicht damit belastet werden. Es sind bei gegebenen Koordinaten und gegebenem Koordinatensystem, systeminterne Berechnungen durchzuführen. So kann sichergestellt werden, dass verschiedene Koordinatensysteme mit geringen Abweichungen in das *WGS84* Format gemäß *EPSG Code* 4326 überführt werden können. Für die Transformation ist es ratsam die Bibliothek PROJ.4 oder die für das Web besser geeignete Proj4js-JavaScript-Bibliothek zu verwenden.

Für die Vorkommen ist eine Vorschaukarte gewünscht. Um eine Karte ohne weitere Konfigurationen, für eine Vorschau zur Verfügung zu stellen, wird eine Standardkarte benötigt. Gemäß der Analyse sind Karten von *Open Street Map* (*OSM*) zwar nicht unbedingt genau, die Lizenz für die Verwendung ist jedoch die

unverfänglichste. Aus diesem Grund wird *OSM* für den Einsatz als Standardkarte bevorzugt. Auch in diesem Einsatzszenario sollte OpenLayers im Sinne der Konsistenz zur Darstellung verwendet werden. Ein weiterer Vorteil besteht darin, das OpenLayers mit der Proj4js-Bibliothek, wenn diese eingebunden ist, zusammenarbeiten kann. In diesem Fall kann die Koordinatentransformation während der Vorschau mit Hilfe der Kombination beider Bibliotheken erfolgen. Die Vorschau dient vor allem zur Kontrolle der Daten durch den Nutzer.

4.2 Backend

Das Joomla!-Backend dient zur Verwaltung von Inhalten, die später im Frontend angezeigt werden sollen. Im Rahmen der Joomla!-Nutzergruppen ist der Zugang zu dem Backend auf spezielle Nutzer beschränkt. Da nur administrative Nutzer mit der Verwaltung der Webseite vertraut sein sollen, ist es sinnvoll alle auf diese Nutzer bezogenen Anforderungen im Backend zu realisieren. Um die einfache Integration der Anwendung in eine Joomla!-Grundinstallation zu ermöglichen, ist es zusätzlich sinnvoll die Anforderungen im Rahmen einer Erweiterung des *CMS* zu implementieren. Hierzu sollte eine Komponente genutzt werden. Komponenten stellen, wie aus der Analyse hervorgehend, eine selbstständige Realisierung dar, die durch eigene Datenbanktabellen unterstützt werden kann.

Zum Aufgabenbereich der Komponente gehört gemäß den Anforderungen, die Erfassung und Verwaltung von Arten und den dazugehörigen Populärnamen sowie der *Taxonomie*, den roten Listen, Vorschaubildern und den verknüpfbaren Metadaten in Form von Spezialisierungen.

Da sich die biologische *Taxonomie* aufgrund von neuen Erkenntnissen verändert, kann diese nicht fest in der Anwendung verankert werden. Aus diesem Grund sind Funktionen nötig, die es ermöglichen die *Taxonomie* zu verwalten und neue Einträge vornehmen zu können, gleichfalls muss es auch möglich sein, Einträge zu löschen oder zu verschieben. In der Verwaltung für die *Taxonomie* sollten zur besseren Nutzbarkeit Drag-And-Drop-Effekte verwendet werden, um die *Taxonomie* zu ordnen. Zusätzlich sollte die Baumstruktur durch die Darstellung verdeutlicht werden. Im Bezug auf die Arten ist auch die Erfassung und Zulassung von Bearbeitern Aufgabe der administrativen Nutzer und damit Teil dieser Komponente.

Da die Mehrsprachigkeit verwaltet werden muss, sollte sie ebenfalls Teil des Backends sein. Bevor eigene Versuche zur Implementierung von Mehrsprachigkeit unternommen werden, sollte jedoch überprüft werden ob es nicht bereits Erweiterungen für die Mehrsprachigkeit von Joomla! gibt. Joom!Fish[1] wäre hier eine Möglichkeit.

[1] http://www.joomfish.net/

Neben der Forderung Vorkommen in Form von „*User Generated Content*“ zu erfassen, soll auch administrativen Nutzern die Erfassung von Vorkommen erlaubt sein. Daher ist die Erfassung der Vorkommen auch Teil des Backends. Administrative Nutzer könnten diese Aufgabe mit Hilfe der im Frontend beschriebenen Funktion durchführen, eine Trennung ist jedoch sinnvoll, da administrative Nutzer auch Daten von anderen Nutzern und für andere Nutzer einfügen dürfen. Alle im Frontend genannten Funktionen bezüglich der Erfassung von Vorkommen sind jedoch im Backend ebenfalls vorhanden. Neben der Erfassung gehört im Backend aber auch die Verwaltung der Vorkommen dazu. Hier ist es wichtig eine Möglichkeit zu schaffen, Daten, die über das Frontend eingefügt wurden, im Backend als solche zu markieren, damit die administrativen Nutzer diese Daten erkennen und prüfen können. Eine Variante dies zu tun, ist die Verwendung des im Joomla!-Framework üblichen Attributes „Published“. Ein Datensatz wäre nicht „published“ solange er nicht durch einen administrativen Nutzer verifiziert wurde.

Neben der Erfassung und Verwaltung von Informationen liegt auch die Konfiguration der angezeigten Karten im Aufgabenbereich der administrativen Nutzer und damit im Backend. Die Komponente benötigt Verwaltungsstrukturen für die Begrenzung einer Karte sowie für die Einbindung von Kartenebenen und deren Zuordnung zu einer Art. Wie aus der Analyse zu erkennen, ist es nur wenig sinnvoll mit eigenen Kartendaten zu arbeiten, da diese den Webserver unnötig belasten würden und die Dynamik einer solchen Karte in den meisten Fällen eingeschränkt bleibt.

Auch wenn die in der Analyse angesprochenen *Tile-Services* meist eine API zur Einbindung der eigenen Karten bieten, widerspricht die Nutzung dieser APIs der Anforderung verschiedene Quellen einbinden zu können. Für die Umsetzung der Konfiguration für Karten sollte daher davon ausgegangen werden, dass zur Anzeige die JavaScript-Bibliothek OpenLayers genutzt wird, da mit dieser Bibliothek diverse Quellen verwendet werden können. Es sollten also die von OpenLayers für Karten und Ebenen benötigten Parameter in der Konfiguration erfasst werden.

Für die Vorkommen ist eine Vorschaukarte gewünscht, dies ist aber auch für die Konfiguration einer Karte sinnvoll. Wie im Frontend sollte hier ebenfalls *OSM* in Kombination mit OpenLayers, im Sinne der Konsistenz, zur Darstellung verwendet werden.

Gemäß den Anforderungen soll eine Möglichkeit vorhanden sein, Bilder hochzuladen, diese Funktionalität kann ebenfalls in der Komponente umgesetzt werden, es ist jedoch zu bedenken das für Joomla! diverse Werkzeuge zur Verwaltung von Bildern zur Verfügung stehen.

4.3 Plugin

Aufgrund der geforderten Darstellung von *Vektor*- oder Verfremdungsdaten, im Weiteren als *Rasterdaten* bezeichnet, kommt es zu dem Problem, dass mindestens zwei Koordinatensysteme abbildbar sein müssen. Obwohl die *Rasterdaten* aus den *WGS84* Daten bei einem Aufruf berechnet werden könnten, hat sich aus Erfahrungen mit der Aufwendigkeit dieser Berechnungen ergeben, dass dies nicht wünschenswert ist. Es soll stattdessen einen zweiten Datensatz in der Datenbank geben der aufgerufen werden kann. Durch die Existenz mehrfacher Datensätze besteht die Notwendigkeit das Kerndatenmodell für die generische Anwendung herauszufiltern und diese extra generierten Datensätze mit Hilfe von Erweiterungen umzusetzen, besonders da das Koordinatensystem für die Verfremdung national abweichen kann. Die angesprochene Dynamik ist am einfachsten mit Hilfe von Plugins zu realisieren, die durch Events in die Speicherung der Vorkommen eingreifen und zusätzlich zu den Koordinaten des Vorkommens eine Verfremdung in der Datenbank ablegen können. Da Plugins auf derselben Ebene wie Backend und Frontend zu finden sind, sind sie in beiden Bereichen bei der Erfassung von Vorkommen einsetzbar.

5 Realisierung

Die angestrebte Lösung ist die Umsetzung einer öffentlich zugänglichen Webanwendung auf Joomla!-Basis, zur Realisierung ist daher ein in Abbildung 5.1 gezeigtes Client-Server Modell nötig, genauer ein Webserver mit installiertem Joomla!-*CMS* und ein Browser als Client. Neben diesen beiden Faktoren ist ein Datenbanksystem vom Typ MySQL zu nutzen, um die dynamische Erzeugung und Darstellung von Inhalten in der Joomla!-Webanwendung zu gewährleisten. Die resultierende Verbindung aus Browser, Webserver und Datenbank, als einzelne Schichten für Präsentation, Logik und *Persistenz*, bildet einen 3-Tier-Aufbau, wie er in Abbildung 5.1 dargestellt wird. Um den Bezug zu diesem Aufbau zu verstärken, sind weitere Beschreibungen in diesem Teil der Arbeit an diese Struktur angelehnt.

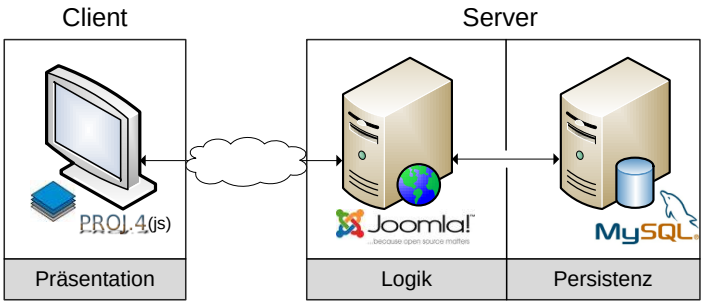

Abbildung 5.1: 3-Tier-Client-Server-Architektur

Die Datenbank als Basis für die darzustellenden Informationen und Ausgangspunkt für jegliche Dynamik der Webanwendung wird im Folgenden zu erst beschrieben. Im zweiten Schritt wird die eigentliche Webanwendung erläutert, zum Einstieg wird auf das bei der Entwicklung vorausgesetzte Nutzerkonzept eingegangen. Es folgen die Erläuterungen des Joomla!-Backends, dessen Aufgaben vor allem mit der Anwendungslogik zusammenhängen, um daraufhin das Frontend behandeln zu können, dessen Schwerpunkt im Bereich der Visualisierung liegt.

In der Beschreibung der Realisierung sind die Worte und deren Wortverwandte „Funktion“ und „Methode“ zu unterscheiden. Wird von einer Funktion gesprochen geschieht dies immer in Bezug auf eine angebotene Funktionalität. Das Wort Methode beschreibt dagegen einen programmierten Algorithmus.

5.1 Datenbank

Die Datenbank ist in Form eines *Entity Relationship Diagramm* (*ERD*) in Abbildung 5.2 zu sehen. Das Datenbankdesign ist allerdings nur zum Teil durch diese Arbeit beeinflusst worden, daher wird es mit dem Ziel die Zusammenhänge zwischen Datenbank und Webanwendung zu verdeutlichen nur eingeschränkt erläutert, ein ausführliches Konzept zur Datenbankstruktur ist nicht Teil dieser Arbeit. Das Datenbankmodell wurde im Rahmen des Projektes, möglichst unabhängig von der Webanwendung, gemeinsam erarbeitet.

Um die Vorteile des Joomla!-Frameworks nutzen zu können, wurde die Datenbank jedoch in einigen Punkten erweitert. Insgesamt drei Attribute sind zur Verwendung durch Joomla! in die Datenbank eingefügt worden. Zur späteren Vermeidung von Überschneidungen, bei der, für ein *CMS* typischen, simultanen Bearbeitung von Inhalten, sind die durch Joomla! vorgegebenen Felder „checked_out_time“ und „checked_out“ in die häufig verwendeten Tabellen eingefügt worden. Mit Hilfe dieser Attribute können vorhandene Funktionen des Frameworks eingesetzt werden, um festzustellen, ob ein Eintrag zum Zugriffszeitpunkt durch Andere bearbeitet wird. Sollte ein Eintrag in Bearbeitung sein, kann der Zugriff einer weiteren Person verhindert werden. Auch das Attribut „published“ ist Joomla!-spezifisch. Mit diesem Attribut wird bestimmt, ob ein Eintrag für die Darstellung im Frontend freigeschaltet ist. Das Framework bietet auch hierfür Funktionen zur Verwaltung an.

Bei dem Entwurf des Modells wurde versucht den Entitäten, soweit möglich, aussagekräftige Namen zu geben. Zur angesprochenen Klärung der Zusammenhänge wird daher nur begrenzt auf das Schema eingegangen, um eventuelle Unklarheiten zu beseitigen und Besonderheiten für die Implementierung hervor zu heben. Die im Folgenden beschriebenen Entitäten stellen das Kerndatenmodell dar. Von diesem Modell ausgehend können Erweiterungen angefügt werden, das vorliegende Modell muss allerdings von diesen Erweiterungen unabhängig bleiben. Wie im *ERD* zu erkennen, gibt es zwei Kerntabellen, deren Attribute durch weitere Tabellen ausgefüllt werden.

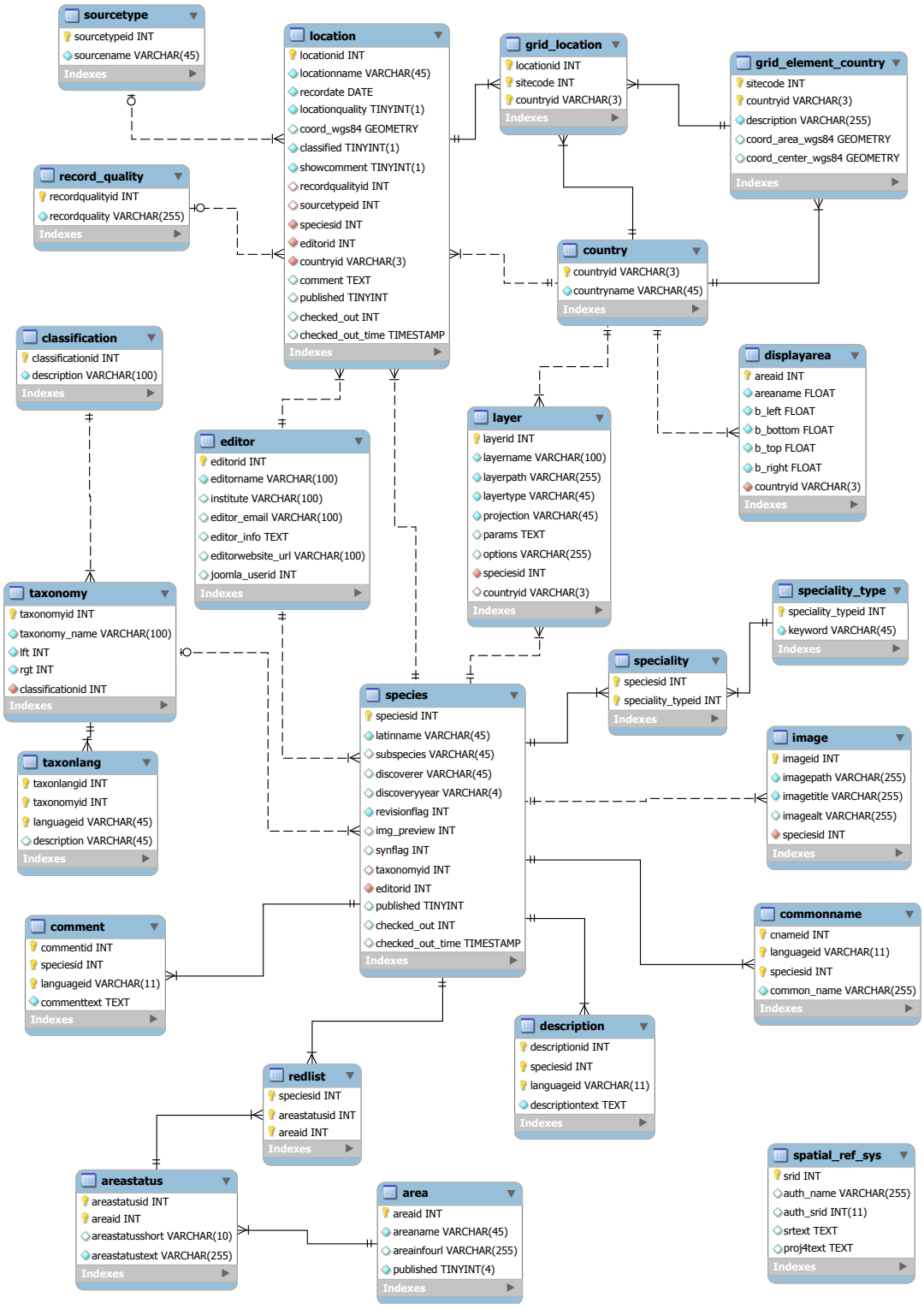

Abbildung 5.2: Entity Relationship Diagramm der verwendeten Datenstruktur

5.1.1 Species

Die erste Kerntabelle ist „species“ sie stellt die Grundlage für die Beschreibung einer biologischen Art dar. Jede biologische Art setzt sich neben textgebundenen Informationen unter anderem aus einem Bearbeiter („editor“) der für diese Art verantwortlich ist, einer Einordnung in die biologische *Taxonomie* („taxonomy“), verschiedenen Eigenschaften („speciality“) und der Einordnung in vorhandene rote Listen zusammen („redlist“). Zusätzlich sind für jede biologische Art, Vorschaubilder („image“) und einzelne Kartenebenen („layer“) erfassbar.

Für jede Art können verschiedene Populärnamen („commonname“), Kommentare („comment“) und Beschreibungen („description“) in unterschiedlichen Sprachen erfasst werden, die Sprachen müssen Joomla! allerdings bekannt sein. Zur Erfassung der Sprache besitzt jede Entität ein Attribut „languageid“ in der ein Sprachschlüssel abgelegt ist. Jeder Sprachschlüssel besteht aus einem Sprachcode gemäß ISO 639-1 im Format Alpha-2. Dieser Sprachcode wird durch einen Bindestrich mit einem Ländercode gemäß ISO 3166-1 ebenfalls im Format Alpha-2 zu dem Sprachschlüssel ergänzt. Die Möglichkeit verschiedene Sprachvarianten zu nutzen ist ein Vorgriff auf zukünftige Entwicklungen im Bezug auf Mehrsprachigkeit. Mehrsprachigkeit ist jedoch nicht Teil dieser Umsetzung.

Die Erfassung einer Art in rote Listen benötigt ebenfalls weitere Tabellen, da die Erfassung an sich nicht so trivial ist, dass dies mit einer einzigen Tabelle („redlist“) gelöst werden könnte. Rote Listen sind in ihrer Zuordnung sehr komplex, da Arten je nach Betrachtungsweise („area“) lokal, regional oder weltweit in unterschiedliche Stadien („areastatus“) eingeordnet werden können. Eine Art kann daher in mehrere Stadien der Bedrohung eingeordnet werden. Sie kann zum Beispiel weltweit als bedroht gelten, weil sie nur noch in einem Land vorkommt, da sie in diesem Land jedoch häufig vorkommt, muss sie auf Landesebene keinesfalls als bedroht eingestuft sein.

Die *Taxonomie* ist ebenfalls ein Sonderfall, der eine komplexere Lösung benötigt, da sich die taxonomische Einordnung durch die Genetik und den damit verbundenen wissenschaftlichen Erkenntnissen im ständigen Wandel befindet. Dieser Wandel und die damit verbundene Dynamik mussten in dem Datenbankentwurf berücksichtigt werden. Hierzu wurde die Methode *Nested Sets* eingesetzt. Mit dieser Methode werden die einzelnen Hierarchiestufen durch Angabe einer linken und rechten Grenze auf einem Zahlenstrahl, wie in Abbildung 5.3 zusehen, ineinander verschachtelt. Das Prinzip der Schachtelung ist vergleichbar mit einer russischen Matroschka, wobei die höhere Hierarchiestufe die niedrigere beinhaltet.

Aus dieser Anordnung lässt sich ein Baum erzeugen, in dem die einzelnen Elemente durch Lösch- und Schiebeoperationen bewegt werden können. Die Klassifikation („classification“) einer Stufe sorgt, unabhängig von ihrer Position, für die

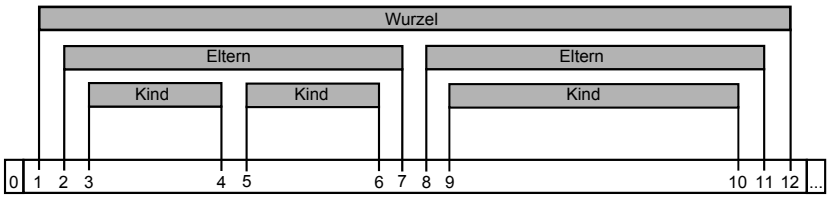

Abbildung 5.3: Struktur der Nested Sets

richtige Benennung der Ebene, in der sich eine Stufe befindet, da die Position, aufgrund verschieden tiefer Zweige des Baumes, keine Auskunft über Ebenen eines Knotens geben kann. Für die Taxonomiebezeichnungen gibt es ebenfalls die Möglichkeit der mehrsprachigen Benennung („taxonlang“).

5.1.2 Location

Die zweite Kerntabelle ist „location“. Durch diese Tabelle sind einzelne Vorkommen erfassbar. Sie ist die Grundvoraussetzung für die Darstellung von geographischen Informationen.

Jedes Vorkommen besitzt Textinformationen und zusätzliche Entitäten. Einem Vorkommen kann die Art der Quelle („sourcetype“), zum Beispiel Literatur, zugeordnet werden. Des Weiteren ist es möglich die Qualität („record_quality“) eines Vorkommens festzulegen. Die Qualität gibt darüber Auskunft, ob ein Vorkommen direkt beobachtet wurde oder ob die Existenz, zum Beispiel durch wissenschaftliche Untersuchungen, indirekt gefolgert worden ist. Zusätzlich kann jedes Vorkommen neben den originalen Koordinaten mit verfremdeten Koordinaten („grid_element_country“) verbunden werden („grid_location“). Da die Verfremdungen länderspezifisch sind, ist zur eindeutigen Zuordnung eine weitere Entität nötig („country“). Mit Hilfe dieser Entität werden auch die Kartendarstellungen („layer“) eingeordnet, Gebietseingrenzung („displayarea“) für die Darstellungen können ebenfalls vorgenommen werden.

5.2 Webanwendung

Gemäß des 3-Tier-Modells und aufgrund der Entwicklung mit Hilfe des Joomla!-Frameworks werden für die Darstellung der Webanwendung, die bei der Analyse von Joomla! beschriebenen Bereiche Backend und Frontend zur Strukturierung der Beschreibung verwendet. Zusätzlich werden die Elemente Komponente, Modul und Plugin unterschieden. Plugins können jedoch nicht generell in Backend oder Frontend eingeteilt werden, da sie für beide Teile zur Verfügung stehen, weshalb sie gesondert betrachtet werden.

Mit Hilfe der Klasse JText aus dem Joomla!-Framework wurde darauf geachtet, das alle statischen Texte der Webanwendung im Hinblick auf zukünftige Entwicklungen mit Hilfe von ini-Dateien mehrsprachig eingesetzt werden können.

Bevor genauer auf die Anwendungsentwicklung eingegangen wird, muss das Nutzermodell betrachtet werden, da es eine Grundlage für weitere Schritte darstellt.

5.2.1 Nutzermodell

Wie im Rahmen des Realisierungskonzeptes angesprochen soll das Nutzermodell von Joomla! für die Implementierung der Anwendung verwendet werden. Abbildung 5.4 zeigt die verwendeten Joomla!-Nutzergruppen in Bezug auf ihre Rechte für die zu realisierende Anwendung. Die Gruppen stellen die Mindestanforderungen an die Gruppenzugehörigkeit eines Nutzers dar. Aus der in Tabelle 3.2 der Analyse von Joomla! dargestellten Gruppenhierarchie kann entnommen werden, welche Gruppen diesbezüglich dieselben Rechte besitzen. Die in Tabelle 3.2 dargestellten Gruppen sind dazu von sehr eingeschränkten Befugnissen zu weitreichenden Befugnissen sortiert. Prinzipiell können jedoch in den Bereichen Backend und Frontend immer höchstens die Funktionen genutzt werden, die auch angeboten werden.

Da ein nicht registrierter Nutzer dem System nicht weiter bekannt ist, wird er durch Joomla! mit dem Status „guest“ versehen. Er hat damit nur Zugriff auf die öffentlichen Teile der Seite.

Für die Nutzer mit eingeschränkten Rechten im Frontend ist in Joomla! eine Login-Funktion integriert, um den Nutzer dem System vorzustellen. Nach dem Login kennt das System somit den Nutzer und ordnet ihn in die ihm zugeordnete Gruppe ein. Da die registrierten Nutzer nicht in der Lage sein sollen anderweitige *CMS* Inhalte außerhalb der zu entwickelnden Webanwendung zu verändern, hierzu zählt zum Beispiel einen Artikel bearbeiten oder veröffentlichen zu können, werden diese Nutzer der Gruppe „Registered“ zugeordnet. Durch diese Zuordnung bekommen sie Zugriff auf einen im Menü ausgewiesenen internen Bereich und die dort dargestellten Teile der Webanwendung, sie sind jedoch nicht in der Lage andere Joomla!-Inhalte zu verändern. Nutzer, die auf das Backend Zugriff haben sollen, um die Anwendung zu administrieren, müssen mindestens der Gruppe „Manager“ zugeordnet sein, dadurch können sie prinzipiell im Backend arbeiten. Die Administration des *CMS* ist ihnen jedoch nicht erlaubt. Hierzu müssen Nutzer einer Administratorengruppe „Administrator“ oder „Super Administrator“ angehören, dies ist aber nur bei wenigen Nutzern sinnvoll. Ein Artbearbeiter benötigt beispielsweise nicht zwingend die Möglichkeit, das *CMS* zu administrieren, er benötigt nur den Zugriff auf die Anwendung um eine Art bearbeiten zu können. Der Zugriff auf die Anwendung ist einem Artbearbeiter allerdings schon als Mitglied der Gruppe „Manager“ durch eine Access Control List gewehrt. Weitere Nutzer-

gruppen sind für die Umsetzung der Webanwendung nicht nötig.

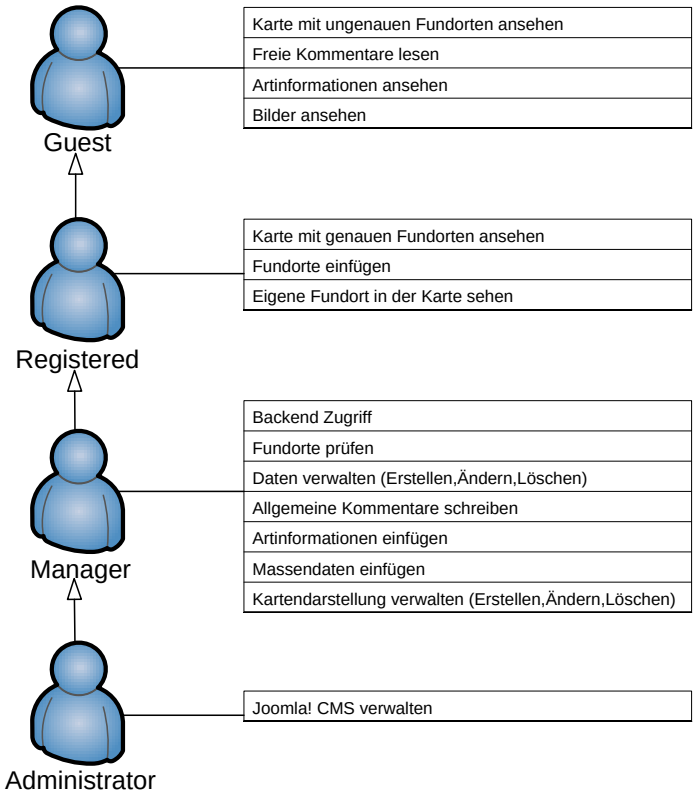

Abbildung 5.4: Für die Komponente verwendete Joomla!-Nutzertypen in Bezug auf ihre Rechte

5.2.2 Backend

Das Backend besitzt vor allem Funktionen, um Daten aus der Datenbank zu verwalten und für die Darstellung im Frontend aufzubereiten. Daher besteht das Backend der zu entwickelnden Anwendung hauptsächlich aus Formularen für die Verwaltung der Daten. Es stellt eine Schnittstelle zur Datenbank dar, mit deren Hilfe Daten ohne SQL-Kenntnisse in die Datenbank eingepflegt werden können. Aufgrund dieser Eigenschaften sind die Anforderungen an das Backend komplex genug, um eine eigene Komponente dafür zu entwickeln.

Für die Struktur der Komponente wurden die Entitäten der Datenbank auf ihre Zusammengehörigkeit untersucht. Aus dieser Untersuchung gingen verschiedene Aufgabenpakete hervor, die jeweils für die Verwaltung eines Teils der Datenbank zuständig sind. Diese Pakete bilden gleichzeitig die Struktur der Anwendung ab. Folgende Aufgabenpakete sind dazu definiert worden, jedes Aufgabenpaket setzt

sich aus einer Primäraufgabe und daraus resultierende Sekundäraufgaben zusammen:

- **Species**, *Primäraufgabe:* Verwaltung der Tabelle „species“. *Sekundäraufgaben:* Verwaltung der Tabellen „description“, „comment“, „commonname“, „Images“, sowie der Zuordnungen „speciality“, „redlist“.
- **Images**, *Primäraufgabe:* Zuordnung von Vorschaubildern zu einer Art, es ist nur über das Arbeitspaket Species zugänglich.
- **Locations**, *Primäraufgabe:* Verwaltung der Tabelle „location“, *Sekundäraufgaben:* Vorschau der eingetragenen Orte, Übernahme von Orten in eine Karte, Umwandlung von Koordinaten in das *WGS84*-System.
- **Editors**, *Primäraufgabe:* Verwaltung der Tabelle „editor“, *Sekundäraufgaben:* Schnellübersicht über die im Verantwortungsbereich liegenden Arten.
- **Taxonomy**, *Primäraufgabe:* Verwaltung der Tabelle „taxonomy“, *Sekundäraufgaben:* Verwaltung der Tabelle „taxonlang“ zur Übersetzung.
- **Red Lists**, *Primäraufgabe:* Verwaltung der Tabelle „area“, *Sekundäraufgaben:* Verwaltung der Tabelle „areastatus“
- **Areas**, *Primäraufgabe:* Verwaltung der Tabelle „displayarea“, *Sekundäraufgaben:* Eingrenzung des Viewports mit Hilfe einer Kartendarstellung.
- **Layers**,*Primäraufgabe:* Verwaltung der Tabelle „layer“, *Sekundäraufgaben:* Vorschau eines Layers in einer Kartendarstellung
- **Basics**, *Primäraufgabe:* Verwaltung der Tabellen mit einfachen Daten, die in verschiedenen Arbeitspaketen genutzt werden, „speciality_type“, „source-type“, „record_quality“, „country“, „classification“

Jedem definierten Aufgabenbereich mit Ausnahme der Bereiche Taxonomy und Basics ist eine Listendarstellung, in der die vorhandenen Datensätze zur Übersicht aufgelistet werden, zugeordnet. Jeder Aufgabenbereich besitzt hingegen eine Formulardarstellung, in der einzelne Einträge aus der Datenbank bearbeitet oder neue Einträge in die Datenbank eingefügt werden können. Abbildung 5.5, zeigt eine Sitemap zur Verdeutlichung dieser Backendstruktur.

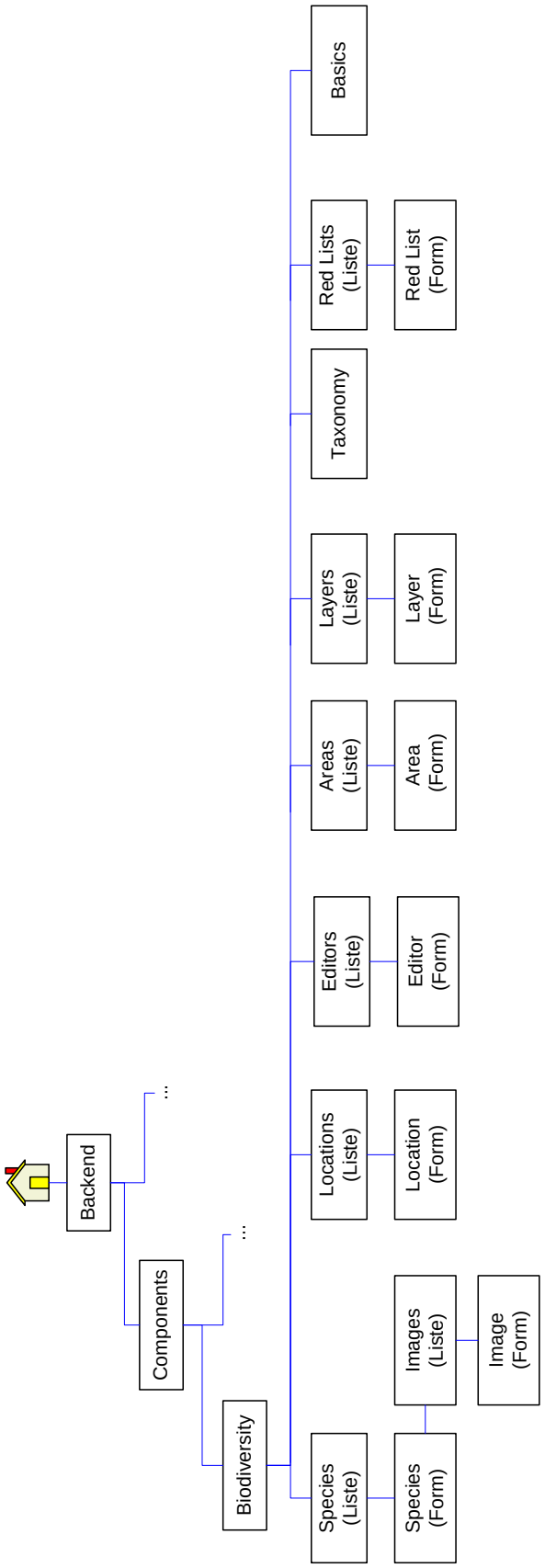

Abbildung 5.5: Sitemap der Komponente im Backend

Architektur

Im Rahmen der Forderung eine möglichst generische Anwendung zu entwickeln, die leicht erweiterbar ist, ist bei der Entwicklung darauf geachtet worden, dass das durch Joomla! vorgegebene *MVC-Muster* eingehalten wird. Zusätzlich wurde jedes Aufgabenpaket weitgehend unabhängig von den anderen Aufgabenpaketen realisiert. Daraus er gibt sich ein zweischichtiger Aufbau.

Die erste Schicht stellt den Zugang zur gesamten Komponente dar, dieser Zugang ist für das *CMS* in der Tabelle „components“ der Joomla!-Datenbank beschrieben. Der Zugang wurde durch ein PHP-Skript realisiert, das eine Routingfunktion besitzt. Anhand des Request-Parameters „controller“ lädt es, sofern vorhanden, ein Controllerobjekt aus den definierten Aufgabenpaketen oder nutzt ein Standardcontrollerobjekt. Alle weiteren Parameter des Request werden an das gewählte Objekt weitergeleitet. Das Controllerobjekt, im weiteren Verlauf Controller genannt, ist die erste Instanz der zweiten Schicht.

Diese zweite Schicht unterteilt sich in die eigentlichen MVC-Bereiche. Jeder Controller greift zur Verarbeitung von Daten auf eine Zahl von Modelobjekten, im Folgenden Model genannt, zurück. Außerdem kennt jeder Controller verschiedene Viewobjekte, weiterführend als View bezeichnet, für die Darstellung der Daten. Der Controller vermittelt zwischen dem Model und der View, während die View Daten von dem Model abrufen kann. Ein Model hat zusätzlich Zugriff auf Tableobjekte, hier Table genannt. Diese Objekte vereinfachen die Nutzung der Datenbank, in dem sie die Struktur einer Datenbanktabelle objektbasiert abbilden und einheitliche Methoden zur Bearbeitung der Datenbanktabelle bereitstellen. Abbildung 5.6 zeigt den Grundaufbau exemplarisch für die gesamte Komponente an der Umsetzung des Aufgabenpaketes Areas.

Areas besitzt als Aufgabenpaket ebenfalls einen Controller. Dieser Controller besitzt neben einem Konstruktor, eine generelle Methode *display()* mit deren Hilfe der Controller die View aufruft, die beim Aufruf des Controllers, ohne weitere Request-Parameter, angezeigt werden soll. Die Methode initialisiert beim Aufruf die nötige View. Im nächsten Schritt wird ein Model initialisiert. Dieses Model wird der View übergeben. In der View wird dann die Methode Display aufgerufen. In der Methode Display der View, werden alle benötigten Daten von dem Model abgefragt. Danach fügt die View die Daten in ein *Template* ein, das daraufhin dargestellt wird.

Des Weiteren besitzt der Controller die Methoden *edit()*, *publish()*, *save()*, *remove()* und *cancel()*. Jede dieser Methoden erfüllt einen sogenannten Task. Der Name der Methode und der als Request-Parameter angegebene Task sind identisch. Im Fall Areas wechselt der Controller mit dem Task „edit“ von der Standardansicht in Form einer Liste aller Areas, zur Formularansicht einer Area, deren Identifi-

kation mit der im Request übermittelten übereinstimmt. Wenn keine oder eine unzureichende Identifikation vorhanden ist, wird ein leeres Formular geladen, das zum Einfügen eines neuen Datensatzes genutzt werden kann.

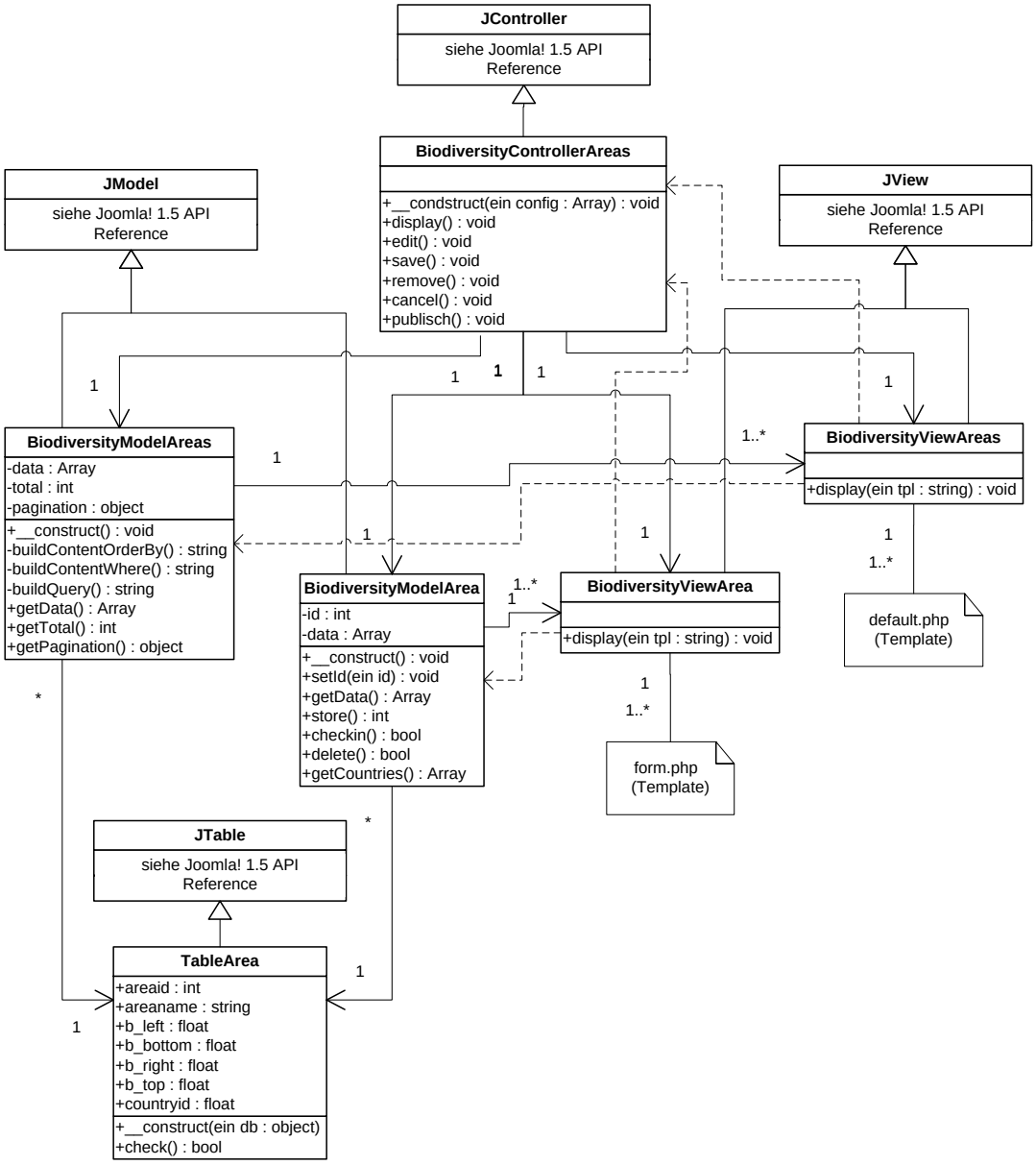

Abbildung 5.6: Klassendiagramm des Aufgabenpaketes Areas, exemplarisch für den Aufbau der anderen Pakete

Zur besseren Übersichtlichkeit im Code und zum Erhalt der festgelegten Benennung von Methoden wird neben einer neuen View auch ein neues Model für die Ansicht geladen. Dadurch wird vermieden, dass sich Benennungen für die Methoden verschiedener Models überschneiden. Während das Model für die Listendarstellung nur die Abfrage von Daten und die Beschränkung der Datenmenge bei einer Abfrage ermöglicht hat, unterstützt das neue Model die für ein Formular nötigen Methoden zur Bearbeitung der darunter liegenden Datenstruktur der Datenbank.

Beispielhaft sind Methoden wie *store()* zum Speichern und *delete()* zum Löschen der Daten. Der Controller nutzt diese Methoden zu Bearbeitung der übermittelten Tasks. Heißt der Task beispielsweise, wie in Abbildung 5.7 am Ende dieses Unterabschnittes dargestellt, „save“ ruft der Controller die Methode *store()* im Model für die Formularansicht auf. Um die Daten in die Datenbank schreiben zu können, kennt dieses Model zusätzlich die Tableobjekte der Datenbanktabellen, mit denen es arbeiten muss. Im Falle von *store()* arbeitet das Model die per Request übermittelten Formulardaten auf und gibt sie an das zuständige Tableobjekt weiter. In jedem Formular wird mit Hilfe der von Joomla! über die Klasse JHTML zur Verfügung gestellten Behavior Formvalidator mittels JavaScript geprüft ob Fehler beim Ausfüllen der Felder gemacht worden sind. Zusätzlich überprüft das Tableobjekt vor der endgültigen Speicherung die Daten mit einer *check()*-Methode auf Vollständigkeit und Integrität, sonst wird die Speicherung abgebrochen und die eingegebenen Daten werden verworfen. Bei einem positiven Ergebnis wird auf Basis des Primärschlüssels einer Tabelle entschieden, ob die übermittelten Daten ein SQL-UPDATE oder ein SQL-INSERT darstellen. Dazu wird eine vordefinierte Methode genutzt in der geprüft wird ob der Primärschlüssel des zu speichernden Datensatzes Null ist oder ob eine Identifikation größer als Null vorhanden ist. Mit Null ist die Ziffer sowie der Wert oder leerer Text gemeint.

Das Vorgehen bei anderen Tasks wird analog oder zumindest ähnlich durchgeführt. Im Fall *remove()* werden einzelne Datensätze anhand der durch den Request übermittelten Identifkationen gelöscht, in diesem Fall bereitet das Model die Daten auf und lässt die Löschung durch ein Tableobjekt erledigen. Diese Methoden sind jedoch nur Teil der Models, die auch in Verbindung mit den Formularansichten eingesetzt werden.

Jedes Model der Anwendung besitzt in den meisten Fällen zumindest eine *get()*-Methode, die von der View zum angesprochenen Laden der darzustellenden Daten für die Einbindung in ein *Template* genutzt wird. Das Area-Model besitzt zwei *get()*-Methoden, eine zur Abfrage der Tabelle „displayarea“ und eine Weitere um Einträge aus der Tabelle „country“ für ein Drop-Down-Feld zu nutzen. Diese *get()*-Methoden benötigen jedoch kein Tableobjekt. Da die einzelnen SQL-SELECT-Anweisungen zum Teil wesentlich komplexer sind und mehrere Tabellen beanspruchen, ist es nicht sinnvoll ein Tableobjekt zur Abfrage zu nutzen.

Verknüpfungen und Sub-SELECTs zwischen Tabellen können beispielsweise nicht durch ein Tabelobjekt abgebildet werden, da nur eine Tabelle pro Table-Klasse nachgebildet werden kann.

Obwohl das Aufgabenpaket Areas weniger Komplex ist, als die meisten anderen Pakete, ist es ausreichend um die Grundstruktur der in dieser Anwendung vorkommenden Pakete zu erläutern. Prinzipiell ist diese Struktur in jedem der Pakete vorhanden, der einzige Unterschied ist die Komplexitäten der Funktionen. Diese Komplexität kommt jedoch größtenteils durch die Sekundärfunktionen zustande. Daher sollen im weiteren Verlauf spezielle Lösungen im sekundären und zum Teil im primären Bereich einzelner Pakete erklärt werden.

In den meisten Fällen sind Probleme in den Bereichen aufgetaucht, in denen heute mit *AJAX* gearbeitet wird. Das bedeutet in den Bereichen, in denen Daten bearbeitet und gespeichert werden, ohne dass sich eine dargestellte Seite vollständig neu lädt. Dies ist zum Beispiel gewollt, um während der Bearbeitung gleichzeitig andere Inhalte zur späteren Verwendung einfügen zu können, ohne den Verlust von Formulardaten nach dem erneuten Laden einer Seite in Kauf nehmen zu müssen. Obwohl Joomla! um eine *AJAX*-Funktionalität erweitert werden kann, ist es bis dato kein Anliegen der Kernentwickler. Aus diesem Grund sollte im Backend, im Konsens dazu, ebenfalls soweit möglich auf *AJAX* im Anwendungskern verzichtet werden. Dafür mussten jedoch andere Lösungen gefunden werden, die in der Nutzbarkeit mit dem Komfort einer *AJAX*-Lösung vergleichbar sind.

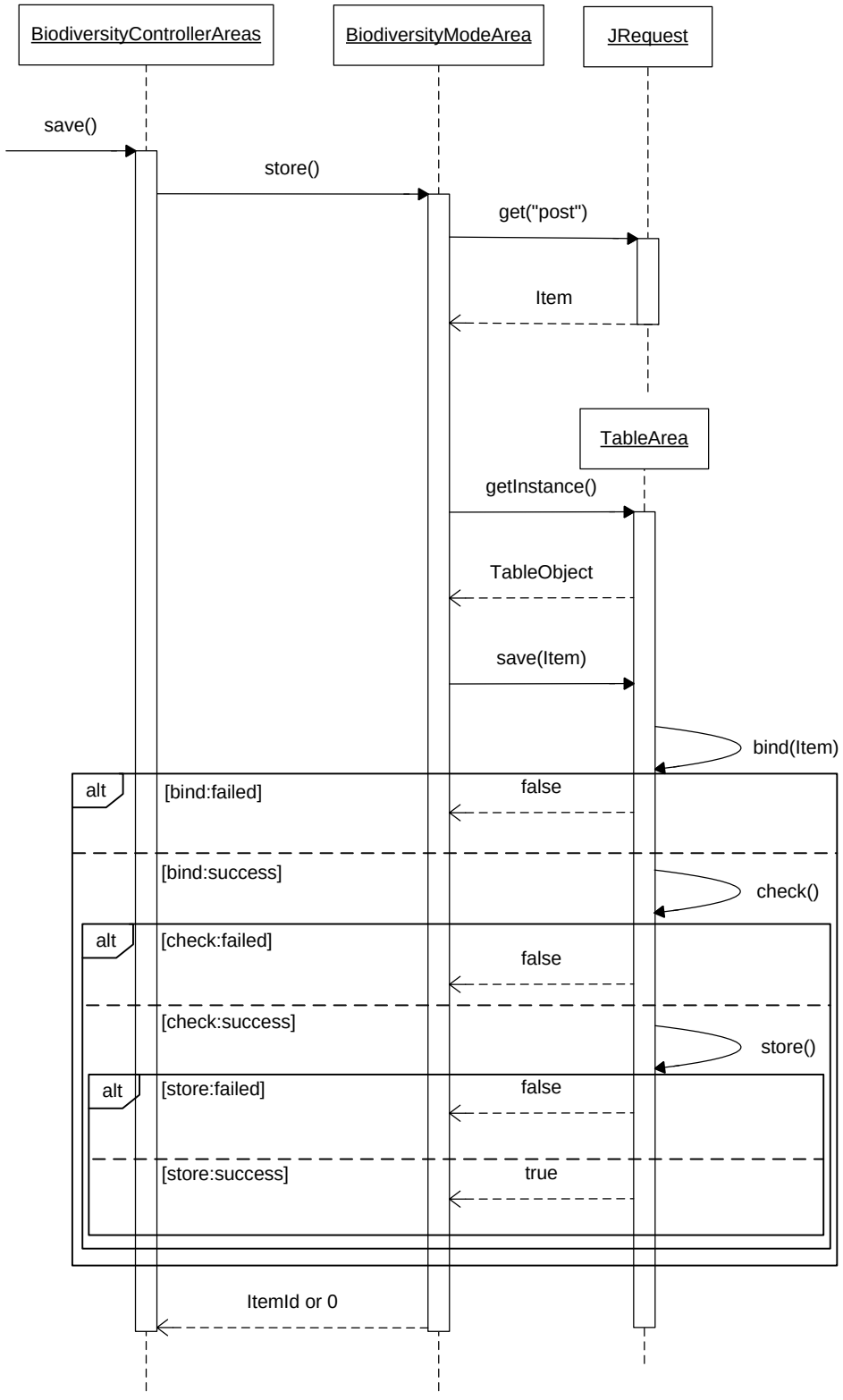

Abbildung 5.7: Sequenzdiagramm für die Speicherung von Formulardaten am Beispiel des Aufgabenpaketes Areas

Taxonomie Verwaltung

Aufgrund der Verwendung von *Nested Sets* für die Speicherung der *Taxonomie*, ergeben sich andere Anforderungen für ein Formular zur Verarbeitung der Daten, als es für eine normale Tabelle üblich ist. Zum einen muss die Darstellung die vorhandene Baumstruktur wiedergeben, während bei Operationen an einzelnen Spalten einer Tabelle gleichzeitig Abhängigkeiten zu Eltern- und folgenden Kindknoten beachtet werden müssen. Für die Implementierung der Darstellung einer Baumstruktur gibt es verschiedene Ansätze.

Ein Ansatz ist die in Joomla! für die Abbildung des Dateisystems genutzte Darstellung, diese Darstellung kann über die JHTML-Klasse des Frameworks als Behavior mit dem Namen *Tree* angesprochen werden und basiert auf der JavaScript-Bibliothek MooTree, die wiederum auf das MooTools Framework zurückzuführen ist. Diese Behavior ist jedoch nur begrenzt interaktiv. Sie bietet Funktionen zum ein- und ausklappen einzelner Äste. Allerdings gibt es keinerlei Funktionen zum Verschieben einzelner Knoten oder ganzer Äste. Das in Joomla! enthaltene JavaScript-Framework MooTools unterstützt aber die Umsetzung eines Drag-And-Drop-Effekts. Um den Tree mit diesem Drag-And-Drop-Effekt auszustatten, wären allerdings direkte Änderungen im Framework nötig, was spätere Versionsupdates des *CMS* erschweren würde.

Eine andere Möglichkeit besteht darin außenstehende Lösungen zu nutzen, Mif.Tree[1] ist eine ebenfalls auf den in Joomla! vorhandenen MooTools aufbauende Lösung, und besitzt unter anderem alle nötigen Funktionen ohne ein weiteres JavaScript-Framework einsetzen zu müssen. Im Fall des Einsatzes von Mif.Tree kommen jedoch neben den vorhandenen Ressourcen weitere zur Komponente hinzu, das erhöht den Speicherplatzverbrauch der Komponente und schafft eine zusätzliche Abhängigkeit.

Aus diesem Grund wurde zur Darstellung der Hierarchie letztendlich die vorhandene *Tree-Behavior* der JHTML-Klasse übernommen. Um trotzdem Drag-And-Drop nutzen zu können, wurde die MooTools-Klasse Drag verwendet. In Abbildung 5.8 ist der Aufbau der Verwaltungsseite für die *Taxonomie* zu sehen.

Die Seite ist in zwei Teile geteilt, auf der linken Seite kann durch die Baumstruktur navigiert werden. Wird ein Knoten ausgewählt, werden auf der rechten Seite neben seinen Attributen die Kindknoten angezeigt, die Kindknoten können per Drag-And-Drop ausgewählt und bewegt werden.

[1] http://mifjs.net/tree/

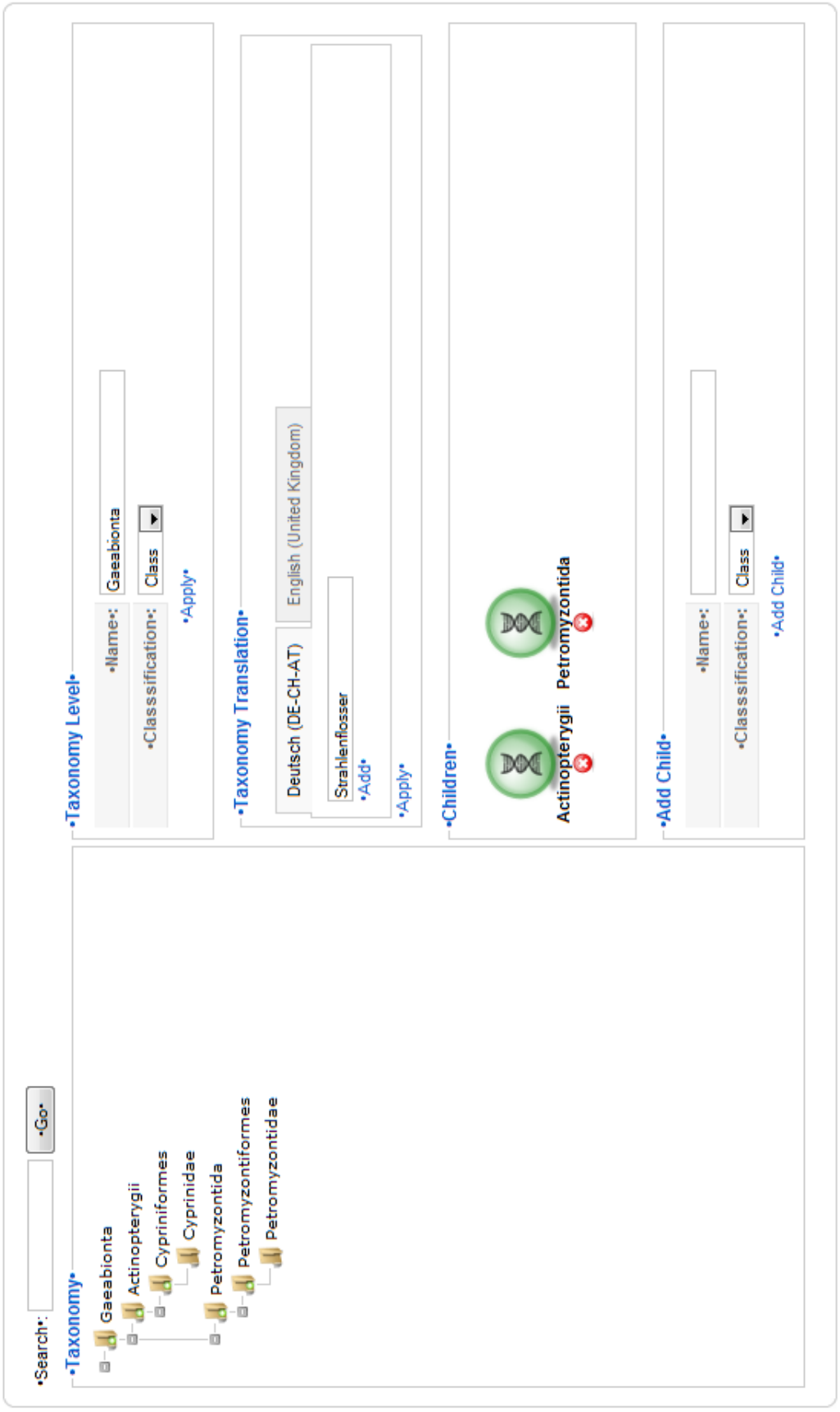

Abbildung 5.8: Verwaltung für die Taxonomie im Backend

Zieht man einen Kindkonten auf einen Anderen, wird im Controller die Methode *move()* aufgerufen. Per Request-Parameter werden dem Model der Zielknoten für die Verschiebung und der zu verschiebende Knoten übergeben. Mit Hilfe des zur *Taxonomie* gehörenden Tableobjekts wird eine simulierte Löschung des zu verschiebenden Knotens durchgeführt. Es wird simuliert, da durch ein SQL-UPDATE alle Knoten so verändert werden, als wäre der Knoten gelöscht worden. Theoretisch wäre es auch möglich eine richtige Löschung vorzunehmen, der Nachteil ist jedoch der Verlust des Wertes für den Primärschlüssel eines Objektes. Spätestens beim Erzeugen des Objektes an anderer Stelle, wäre dieser nicht mehr der Selbe. Das führt dazu, dass alle mit diesem Objekt verknüpften Tabellen aktualisiert werden müssten. Je nach Menge der Datensätze stellt dies einen höheren Aufwand dar, als die Simulation und damit der Erhalt der Primärschlüssel.

Im nächsten Schritt werden alle Kindknoten des zu verschiebenden Knotens durch das Model mit Hilfe des Tableobjektes durch Löschen der Links- und Rechtswerte als ebenfalls zu verschieben markiert.

Zuletzt wird der zu verschiebende Knoten an neuer Stelle eingefügt, in dem er zwischen die Links- und Rechtswerte des Zielknotens einsortiert wird, die mit Null markierten Kindknoten folgen rekursiv. Dabei wird der komplette Teilbaum bei jeder Einfügeoperation um den Wert zwei auf dem Zahlenstrahl erweitert. Abbildung 5.9 zeigt ein Sequenzdiagramm des beschriebenen Ablaufs.

Mit Hilfe der *move()*-Methode wird es ermöglicht, Objekte ineinander zu verschieben. Dies führt dazu, dass Objekte zumindest untergeordnet werden können. Mithilfe der Funktion einzelne Knoten aus dem gesamten Baum zu entfernen und der damit verknüpften Verschiebung aller Kindknoten um eine Ebene aufwärts ist es zusätzlich möglich, Knoten nach oben zu verschieben und von dort aus neu einzuordnen.

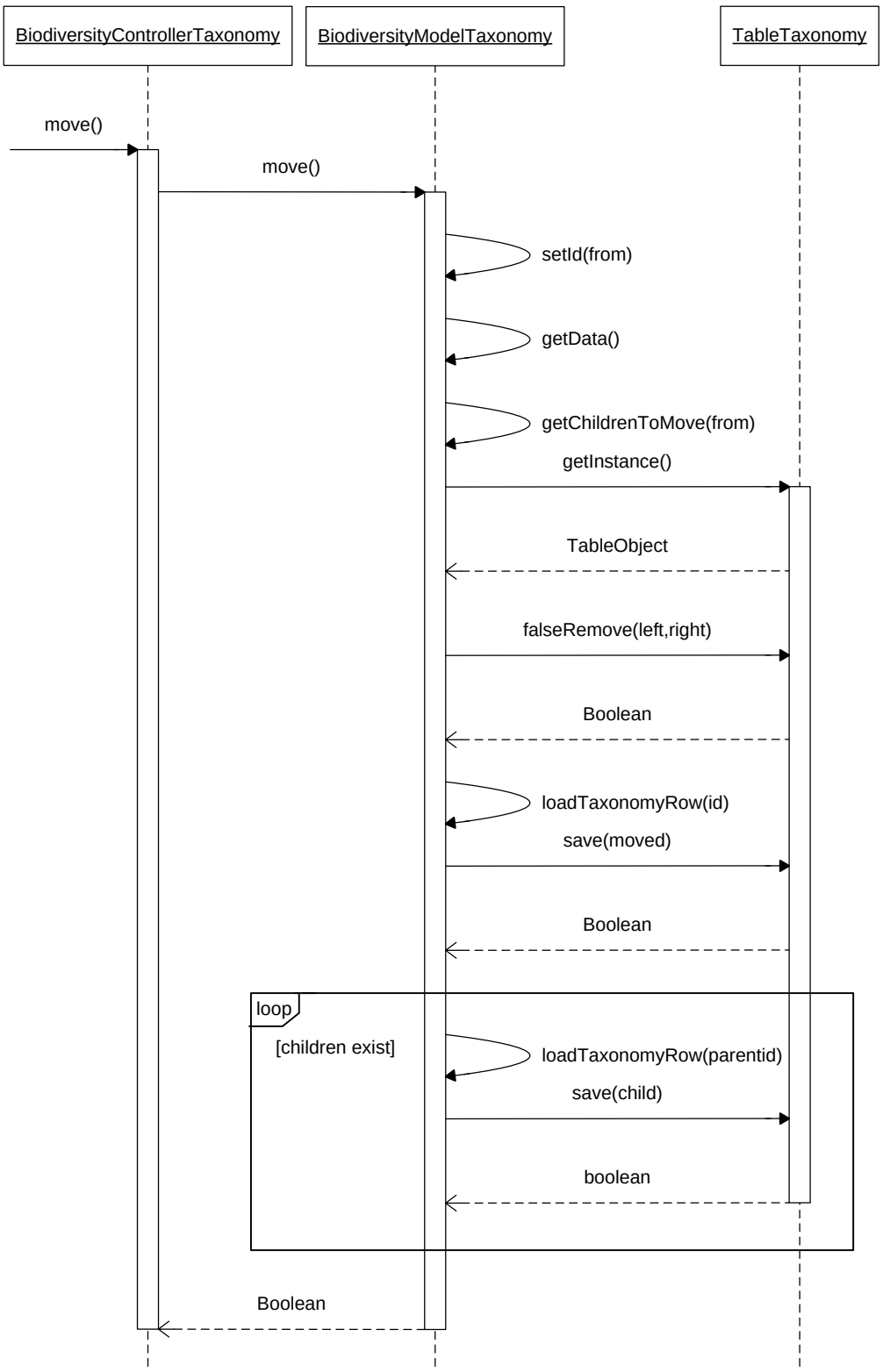

Abbildung 5.9: Sequenzdiagramm der Move-Funktion

Einbindung von Vorschaukarten

Im Gegensatz zur Darstellung der *Taxonomie* kann für die Kartendarstellung nicht auf die Einbindung externer Hilfsmittel verzichtet werden. Da die Kartendarstellung ein umfangreiches Thema ist, zu dem verschiedene Lösungswege vorhanden sind, sollte davon abgesehen werden eigene JavaScript-Lösung zu implementieren, da in jedem Fall eine Lösung entstehen würde die weniger ausgereift ist. Daher ist die Kartendarstellung mit der, in der Analyse von Methoden und Technologien angesprochenen, JavaScript-Bibliothek OpenLayers eingebunden worden.

Für die Kartendarstellung gibt es im Backend drei Schwerpunkte, zum einen die Vorschau eines eingetragenen Punktes vor der Speicherung für das Aufgabenpaket Locations. Zum anderen gibt es die Vorschau von definierten Kartengrenzen für das Aufgabenpaket Areas und die Vorschau einer einzubindenden Kartendarstellung für das Frontend im Aufgabenpaket Layers. In jedem dieser Aufgabenpakete ist, auf der jeweiligen Formularseite, die OpenLayers-Bibliothek und JavaScript zur Initialisierung der Karte und zur Bereitstellung der nötigen Interaktionsfunktionen eingebunden. Die Vorschau von Punkten und Bereichen ist im Ansatz ähnlich. In beiden Fällen werden die *OSM Tile-Services* Osmarender und Mapnik eingebunden, sie dienen als Hintergrund und Bezug für die Vorschau.

Zur Eintragung von Koordinaten für ein Vorkommen sind zwei Felder vorgesehen, in diese Felder können die Koordinaten eines Koordinatenpaares, das heißt Werte für geografische Länge und Breite, in dezimaler Form eingetragen werden. Bei einem neuen Eintrag in ein vorerst leeres Formular ist zusätzlich eine Auswahlliste mit *Koordinatenreferenzsystemen* vorhanden. Sind die Koordinaten nicht im *WGS84*-Format, kann ein anderes Format eingestellt werden. Als Wert der jeweiligen Auswahloption sind die Parameter für die PROJ.4-Bibliothek gespeichert. Da die, ebenfalls unter „Aktuelle Methoden und Techniken“ angesprochene, JavaScript-Bibliothek Proj4js mit denselben Parametern genutzt werden kann, ist dies der Bezugspunkt für die benötigten Parameter, um die Koordinaten des entsprechenden Systems in das in der Karte genutzte System umwandeln zu können. Dazu ist in der Formularansicht von Locations zusätzlich die Proj4js-JavaScript-Bibliothek eingebunden. Nachdem „Preview“ ausgewählt wurde, werden die eingetragenen Koordinaten umgerechnet und in die Karte als Vektorgrafikpunkt übernommen. Neben dem Eintragen von Koordinaten besitzt die Karte im Locations-Formular zusätzlich einen Handler für Mausklicks. Nutzern wird dadurch die Möglichkeit gegeben, einen Punkt per Klick in die Karte auszuwählen, woraufhin die Koordinaten in die dafür vorgesehenen Felder eingetragen werden.

Die Vorschau im Aufgabenpaket Areas benötigt diese Funktion nicht, stattdessen kann der User die Karte zurecht schieben und die Grenzen in der Form Links/Unten und Rechts/Oben mit einem Klick auf einen Button, in die dafür vorgesehenen Felder übernehmen. In diesem Formular gibt es auch keine Um-

rechnungsfunktionen, da zum einen davon ausgegangen wird, dass ein Nutzer die angesprochene Bedienung nutzt. Zum anderen besteht seltener die Möglichkeit, dass Begrenzungen aus anderen Quellen, mit anderen Koordinatensystemen, ermittelt werden. Dies wird bei den Punkten der Vorkommen häufiger der Fall sein, da in diesem Zusammenhang mehr Einträge existieren.

Die Vorschau der Layers funktioniert auf andere Weise, zwar ist ebenfalls die OpenLayers-Bibliothek und weiteres JavaScript eingebunden, allerdings ist es nicht möglich, Kartendarstellungen in Form von Kartenebenen mit Hilfe von Parametern für den Typ zu erzeugen. Da OpenLayers klassenbasiert ist, müssen für jeden Ebenentyp einer Karte Klassen genutzt werden. Diese Klassen benötigen wiederum Parameter. Aus diesem Grund wird beim Auswählen von Preview eine Switch-Anweisung aktiviert, die den, im dafür vorgesehenen Feld, angegebenen Ebenentyp vergleicht und basierend auf diesem Typ die richtige Klasse auswählt. Zur korrekten Instanziierung der Klasse werden alle weiteren Felder so sortiert, dass sie zu den Parametern des Konstruktors für die Klasse der Kartenebene passen, daraufhin wird eine neue Kartenebene in das OpenLayers Kartenelement eingefügt.

Im Rahmen dieser Umsetzung ist nicht jede Eventualität abgedeckt, dies ist jedoch auch nicht Teil der Arbeit, da es in dieser Arbeit nicht um die vollständige Integration von OpenLayers geht. Viel mehr ist mit der Lösung jede Eventualität abgedeckt, die im Zusammenhang mit der Erfassung und Darstellung von Biodiversitätsdaten, mit Hilfe eines Geoinformationsystems, im Bearbeitungszeitraum benötigt worden ist.

Verwaltung von Preview Images

Das Aufgabenpaket bildet in sofern einen Sonderfall, als es wie in der Sitemap, in Abbildung 5.5, zu erkennen nicht auf derselben Höhe angeordnet ist wie alle anderen Aufgabenpakete. Während die anderen Aufgabenpakete direkt der ersten Darstellungsebene zugeordnet sind, ist Images unter der Formulardarstellung der Species eingeordnet. Die höhere Einordnung wäre ebenfalls möglich gewesen, jedoch bildet dies nicht das Bedienverhalten eines Nutzers ab, wie es bei der Realisierung angedacht war. Während der Diskussion um die Verwaltung der Vorschaubilder ließ die zukünftige Redaktion des geplanten Atlas erkennen, dass es vor allem möglich sein soll neue Bilder zu erstellen oder alte zu modifizieren. Diese Anforderung führte dazu, dass mehrere Bilder einer Art zugeordnet werden können, jedoch immer nur ein Bild als Vorschaubild einer Art angezeigt wird. Wäre die Images Verwaltung höher, also neben den anderen Paketen eingeordnet, wären aufgrund der Datenbankstruktur zwei Schritte zur Bearbeitung nötig. Der erste Schritt ist die Erzeugung oder Veränderung eines Bildes mit Hilfe der Formularansicht für Images. Im zweiten Schritt müsste in die Formularansicht der biologischen Art gewechselt werden, um ein Bild als Vorschaubild zu zuordnen. Die jetzige Realisierung bietet eine einfachere Handhabung. Über einen Link

in der Formularansicht einer Art kann bei Bedarf direkt in eine modale Ansicht zur Bildverwaltung gewechselt werden, um dort ein Bild aus einer Listenansicht auszuwählen oder ein Bild mit dem Wechsel in die Formularansicht zu editieren, beziehungsweise zu erzeugen.

Trotzdem weist das Paket dieselben Strukturen auf, wie auch die anderen Aufgabenpakete. Da es zwei Views gibt, eine Listenansicht und eine Formularansicht, gibt es zwei Models und, um das in der Analyse von Joomla! angesprochene *MVC-Muster* vollständig umzusetzen, zumindest einen Controller. Der Unterschied zu den anderen Komponenten besteht lediglich darin, dass diese Komponente mit dem Parameter tmpl=component im Artformular aufgerufen wird, dadurch wird ihr nicht der für das Backend übliche Header angefügt, Abbildung 5.10 und 5.11 verdeutlichen dies.

Abbildung 5.10: Bilderverwaltung mit Seitenkopf

Mit dem Verlust dieses Headers fehlen auch nötige Schaltflächen zur Navigation und Bearbeitung, dafür besitzt jede View des Paktes abweichend zu den Views der anderen Aufgabenpakete extra eingefügte Buttons zur Steuerung.

Abbildung 5.11: Bilderverwaltung mit Parameter ohne Seitenkopf

Ergänzend benötigt dieser Teil der Komponente einen durch den Nutzer festgelegten Parameter, der global in der Komponente eingetragen werden kann. Dieser Parameter bestimmt den Namen des Ordners, in dem sich die Ordnerstruktur für die vorhandenen Vorschaubilder befindet. Mithilfe dieses Parameters durchsucht das zum Paket gehörende Model alle Ordner unterhalb des bezeichneten Ordners nach Bildern. Die gefundenen Bilder können daraufhin in der Formularansicht für die Zuordnung zu einer Art ausgewählt werden.

Die modale Darstellung wird durch die in Joomla! vorhandene *Modal-Behavior* der, im Rahmen der Taxonomy angesprochenen, JHTML-Klasse erzeugt. Bei dem Aufruf eines modalen Dialoges für die Vorschaubilder wird die Listenansicht mit Hilfe eines *iFrame* dargestellt.

Um eine Bildauswahl möglich zu machen, besitzt jeder Listeneintrag einen Button, der beim Aktivieren eine JavaScript-Methode aufruft. Diese JavaScript-Methode aktiviert mit dem Schlüsselwort „parent“ eine weitere JavaScript-Methode in der ausgeblendeten Formularansicht der derzeitig bearbeiteten Art. Sie übermittelt die Identifikation eines Bildes und einen Bildpfad an das Formular der Art. Die JavaScript-Methode im Artformular ordnet diese Informationen den dafür vorgesehenen Formularelementen zu und aktualisiert das im Formular angezeigte Vorschaubild. Abbildung 5.12 zeigt den Ablauf aus der Sicht eines Bearbeiters.

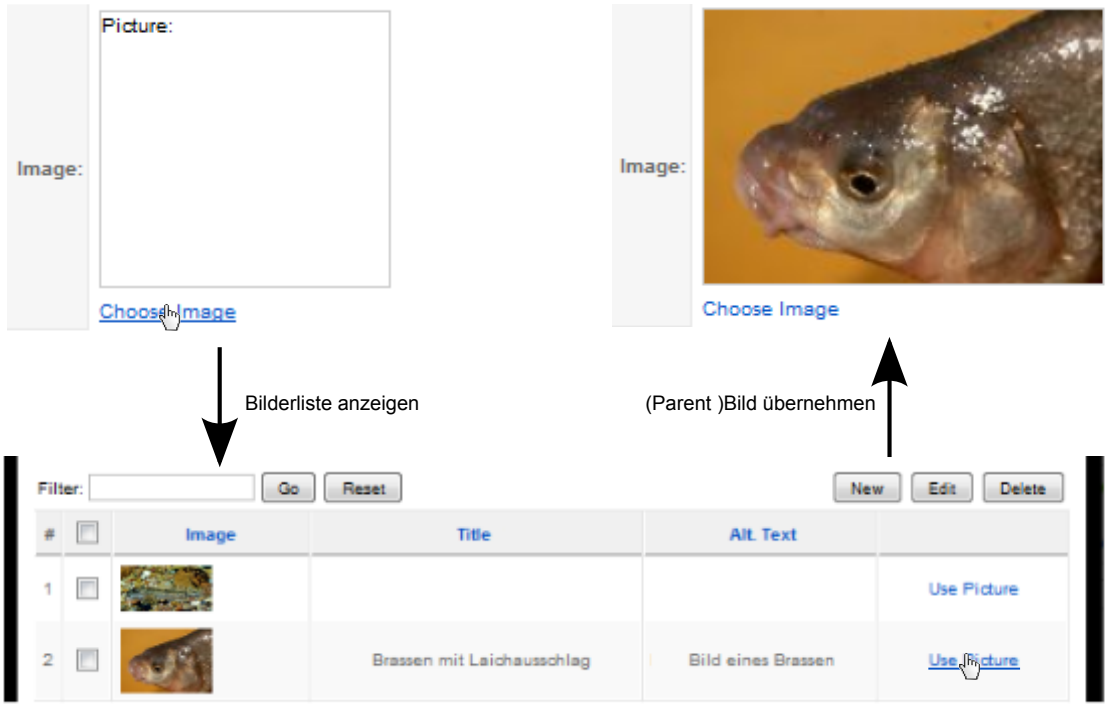

Abbildung 5.12: Zuweisung eines Vorschaubildes zu einer Art aus Sicht eines Bearbeiters mit Fischabbildungen von Prof. Dr. Heiko Brunken

Das Schlüsselwort „parent“ ist nötig, da der *iFrame*, auch wenn die Darstellung dies nicht vermuten lässt, bei dem Aufruf von Images in die Formularansicht der Art eingebettet wird. Mit „parent“ ist daher eine einfache Übertragung von Informationen zwischen den zwei eigenständigen Darstellungen möglich. Da erneutes Laden in einem *iFrame* nicht die Seite betrifft, in die der *iFrame* eingebettet ist, kann beliebig mit den Bildern gearbeitet werden, ohne ungespeicherte Formulardaten des Artformulars zu verlieren. Es ist also ein Ersatz für eine *AJAX*-Lösung.

Verwaltung der Basics

Basics stellen, wie bei der Definition der Aufgabenpakete angesprochen, alle Tabellen dar, die für die anderen Pakete benötigt werden. Basics sind längerfristige Daten und werden meistens nur einmal definiert, um Werte für die Haupttabellen „location“ und „species“ zur Verfügung zu stellen. Sie werden in den meisten Anwendungsszenarien höchstens durch neue Datensätze ergänzt. Daher gibt es für Basics nur die Operationen Löschen und Erstellen, sie können nicht editiert werden, wie es bei den komplexeren Tabellen der Anwendung der Fall ist. Aus diesem Grund werden Basics nicht wie Images einem anderen Aufgabenpaket untergeordnet, sondern zentral administriert. Auch wenn diese Möglichkeit ebenfalls denkbar wäre, erscheint eine zentrale Positionierung sinnvoller, da sonst eine unnötige Komplexität geschaffen würde, die sich negativ auf die *Performanz* auswirken könnte.

Es gibt, wie für die *Taxonomie*, bei den Basics keine Listenansicht, da die Menge der zu verwaltenden Datensätze überschaubar bleibt. Es sind schließlich nur ergänzende Werte, die wenigen Veränderungen unterliegen. Bei den Basics ist die Auflistung daher direkt mit einem Formular verknüpft, das die Operationen Löschen und Einfügen unterstützt. Da die Basics nur eine View besitzen, besitzen sie auch nur ein Model und einen Controller. Um alle Daten in die View zu laden, unterscheidet sich das Basics-Model von dem beschriebenen Standardmodel durch besonders viele *get()*-Methoden. Sie ermöglichen die Erfassung der nötigen Daten und deren Bereitstellung und für die View. Auch die Durchführung von Tasks weicht von dem Standardmodel ab. Bei der Ausführung eines Tasks wird im Model anhand eines Postfix zu einem Task und einer per Request übertragenen Identifikation entschieden, für welches Basics-Element der Task bestimmt ist. Daraufhin werden die entsprechenden Tableobjekte für die Ausführung der Operation angesprochen. Wie üblich besitzt jede in Basics verwaltete Tabelle ein eigenes Tableobjekt.

5.2.3 Frontend

Während das Backend dazu genutzt wird, Daten aus der Datenbank zu verwalten und für die Darstellung aufzubereiten, wird das Frontend vor allem für die Visualisierung der Daten benötigt. Dem Frontend sind dafür zwei Aufgaben zugeteilt. Die

erste Aufgabe besteht in der Anzeige, der in den Anforderungen angesprochenen Daten, mit den dazugehörigen Funktionen zur Verbesserung der Nutzbarkeit. Die zweite Aufgabe ist die Bereitstellung eines Formulars in einem internen Bereich, um den registrierten Nutzern eine Möglichkeit zu eröffnen, die Vorkommen einer Art erfassen zu können.

Architektur

Die Architektur der Anwendung im Frontend unterscheidet sich nur in wenigen Teilen vom Backend, aufgrund der im Vergleich zum Backend geringeren Anforderungen ist sie vor allem weniger kompliziert. Die geringer Komplexität ist in der in Abbildung 5.13 dargestellten Sitemap zu erkennen. Der Aufbau basiert dennoch auf dem *MVC-Muster* und stellt eine Komponente dar.

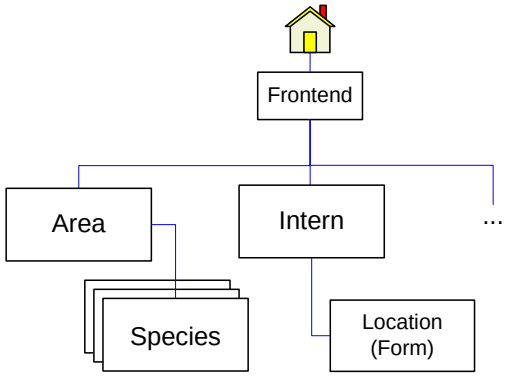

Abbildung 5.13: Sitemap der Komponente im Frontend

Die Entscheidung ebenfalls eine Komponente zu entwickeln ist dadurch begründet, dass durch eine Komponente eine eindeutige Zuordnung zu der Backend-Komponente durch die gleiche Benennung möglich ist und die einzelnen Aufgaben des Frontends, wie auch im Backend, in einer zusammenhängenden Struktur realisiert werden können. Ferner gibt es nur im Fall einer Komponente die Möglichkeit den definierten Content Bereich des Joomla!-Frontends in Anspruch zu nehmen und die Größe der Anzeige, in den Grenzen des *Templates*, selbst zu bestimmen. Da Plugins normalerweise keinen eigenen Anzeigebereich haben und Module nicht zwingend Einfluss auf den für sie vorgesehenen Platz besitzen, ist eine Komponente die beste Wahl, um mit Blick auf die Anforderungen eine möglichst große Karte darstellen zu können.

Diese Wahl bedeutet jedoch nicht, dass beide Komponenten zueinander gleich sind. Ein erster Unterschied besteht in der Einbindung der einzelnen Komponenten in das Joomla!-*CMS*. Während die Backend-Komponente über die Datenbank in das Backendmenü eingefügt wird, wird die Darstellung einer Frontend-

Komponente über den im Backend zu findenden Menümanager für die im Frontend dargestellte Navigation, in das Frontendmenü eingefügt. Zur Erfassung der vorhandenen Darstellungen einer Frontend-Komponente sind standardisierte XML-Dateien nötig. Auf den Inhalt dieser XML-Dateien wird im Rahmen der folgenden Erläuterung zur Struktur noch einmal genauer eingegangen. Zum besseren Verständnis soll jedoch zu nächst die grundlegende Charakteristik der Struktur dargestellt werden.

Aus den, in den Anforderungen für diese Arbeit dargestellten Aufgaben, können zwei zum größten Teil voneinander unabhängige Aufgabenpakete mit Primäraufgabe und verschiedenen Sekundäraufgaben zusammengefasst werden:

- **Location**, *Primäraufgabe:* Verwaltung der Tabelle „location“, *Sekundäraufgaben:* Vorschau der eingetragenen Orte, Übernahme von Orten aus einer Karte, Umwandlung von Koordinaten in das *WGS84*-System. (Identisch zum Backend)

- **Species**, *Primäraufgabe:* Darstellung aller Vorkommen einer Art mit Hilfe einer Karte. *Sekundäraufgaben:* Darstellung weiterer Artinformationen, Aufbereitung der eingetragenen Arten in ein mehrsprachiges Menü mit Filteroptionen, Ortssuche für die Kartendarstellung

Der Aufbau der Komponente ist mit dem Backend vergleichbar. Er basiert ebenfalls auf zwei Schichten, wobei die erste Schicht auch durch ein PHP-Skript mit Routingfunktionen realisiert wird. Beim Aufruf der Komponente lädt dieses Skript das im Request-Parameter „controller“ angegebene Controllerobjekt und leitet alle Request-Parameter an dieses Objekt weiter. Sollte das gesuchte Objekt nicht vorhanden sein, wird ein Standardcontrollerobjekt geladen. Das Controllerobjekt, im weiteren Verlauf Controller genannt, ist wie im Backend die erste Instanz der zweiten Schicht.

Diese zweite Schicht unterteilt sich wiederum in die eigentlichen MVC-Bereiche, die auch im Backend angesprochen werden. Der Unterschied zum Backend besteht darin, dass aufgrund der geringen Menge an Aufgaben sowie der fehlenden Komplexität kein Grund für die Verwendung mehrerer Controller besteht, daher wurde im Frontend nur ein Controller realisiert. Dieser Controller wird gleichzeitig als *Fallback-Controller* eingesetzt, stellt also auch die Realisierung des Standardcontrollers dar. Die weitere Struktur setzt sich aus einer Vermittlungsfähigkeit des Controllers zwischen verschiedenen Modelobjekten, als Models bezeichnet, und den Viewobjekten, als Views bezeichnet, zusammen. Da im Frontend allerdings nur an einem Punkt eine Datenverwaltung stattfindet, ist die Ausprägung dieser Struktur weniger vielschichtig. Wie im Backend werden Views und Models durch die jeweiligen Aufgabenbereiche bestimmt, jedoch besitzt jeder Aufgabenbereich nur eine View und nur ein Model. Dem Controller sind die Views, der

beiden Aufgabenbereiche sowie die zur Bearbeitung nötigen Models bekannt. Wie üblich besitzt der Controller neben einem Konstruktor eine allgemeine *display()*-Methode. In diesem Fall ist sie aber nicht weiter definiert, es wird stattdessen die Methode der Elternklasse aufgerufen. Eine explizite Definition einer View, die standardmäßig beim Zugriff auf den Controller angezeigt wird, wie es in einer *display()*-Methode im Backend der Fall ist, ist nicht nötig, da die View im Rahmen des schon erwähnten Menümanagers festgelegt wird.

In der Frontend-Komponente besitzt jede View eine XML-Beschreibung, in der beschreibende Informationen zu der View abgebildet und Parameter für die View definiert sind. Diese XML-Dateien werden über das Framework automatisch erfasst und im Menümanager ausgewertet. Jede durch eine XML-Datei beschriebene View ist über den Menümanager auswählbar und in Form eines Links im Frontend darstellbar. Der Link besitzt an dieser Stelle alle Informationen in Form von Request-Parametern, die im Backend erst durch die *display()*-Methode definiert werden müssen. Es ist lediglich die Standardmethode der Elternklasse im Frontend nötig um die Anzeige zu initialisieren, da diese Methode anhand der Request-Parameter des Links alle benötigten Teile der Darstellung ableiten kann. Abbildung 5.14 zeigt einen solchen Request und die Bedeutung der einzelnen Parameter.

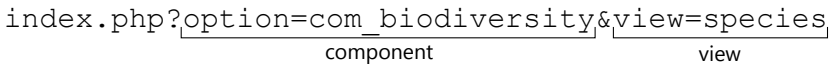

Abbildung 5.14: URL zum Aufruf der Komponente im Frontend

Zusätzlich besitzt der Controller zwei Methoden, um die durch die Views der Aufgabenpakete verwendeten Task zu bearbeiten. Die eine Methode ist *show()* und bezieht sich auf die View des Paketes Species, die andere Methode ist *save()* und bezieht sich auf Location. Es ist noch eine private Methode *sendNotification()* vorhanden. Diese Methode realisiert die in den Anforderungen gewünschte automatische Benachrichtigung per E-Mail, nachdem ein Vorkommen erfasst und gespeichert wurde. Auf die Methoden *show()* und *save()* wird in den folgenden Erläuterungen zu den beiden Views weiter eingegangen.

Species View

Die View kann über den Menümanager ausgewählt werden, um sie in die Navigation im Frontend einzufügen. Zu ihrer Konfiguration sind vier Parameter vorhanden, die in der zur View gehörenden XML-Datei definiert sind.

Der erste Parameter dient zur Auswahl der, beim ersten Aufruf der View genutzten, Zoomeinstellung für die dargestellte Karte. Die auswählbaren Werte werden aus der Tabelle „displayarea“ bezogen. Diese Einstellung sorgt ebenfalls für eine

räumliche Einschränkung in der Darstellung der Vorkommen. Da ein Eintrag in der Tabelle „displayarea“ immer einen Gültigkeitsbereich definiert, werden nur die Vorkommen im Frontend gerendert, die auch in dem zu dem Parameter gehörenden Bereich liegen. Zusätzlich begrenzt die Einstellung die Menge der zu rendernden Layer auf solche mit derselben Länderzuordnung. Ohne diese Zuordnung würden auch Daten geladen werden, die nicht Teil des Kartenausschnittes sind. Das Rendern überflüssiger Daten verbraucht jedoch unnötig viele Ressourcen und würde zu einer geringeren *Performanz* führen.

Der zweite Parameter legt fest, ob die Grundsprache des *CMS* für die mehrsprachige Arten-Auswahl genutzt werden soll. Dies ist insofern sinnvoll, da die Grundsprache nicht zwangsläufig im *CMS* verwendet werden muss, die Artenauswahl jedoch durch die im *CMS* bekannten Sprachen erzeugt wird. Sollte die Grundsprache nicht verwendet werden, würde die Arten-Auswahl an dieser Stelle leer bleiben und für Verwirrung beim Betrachter führen.

Der dritte Eintrag bestimmt, ob alle Namen aus der Tabelle „commonname“ in der Auswahlliste verwendet werden sollen. Da jede Art mehrere Populärnamen besitzen darf, kann die Anzahl der Einträge in der Tabelle „commonname“ größer sein als die Anzahl der tatsächlichen Arten, daher ist es nicht ausgeschlossen, das die Menge aller Namen zu einer unübersichtlichen Navigation führen kann, was wiederum die Nutzbarkeit einschränken würde. Aus diesem Grund kann die Anzeige der Namen mit der beschriebenen Einstellung auf jeweils einen Namen pro Art begrenzt werden. Es wird, in diesem Fall immer der Name einer Spezies für die Anzeige gewählt, der die kleinste Identifikationsnummer besitzt.

Der letzte Eintrag dient dazu, festzulegen, ob ein Verdrängungsmechanismus vorhanden ist und für die Darstellung genutzt werden soll oder ob dies nicht der Fall ist. Sollte keine Verfremdung möglich sein, werden im Zweifel alle Daten die eine Verfremdung benötigen fehlerhaft angezeigt. Durch Ausschalten der Verfremdung werden alle Daten wieder korrekt aber auch genau, angezeigt. Diese Option ist besonders hilfreich, falls ein Einsatz ohne Verfremdung geplant ist.

Die in Abbildung 5.15 zusehende View des Paketes Species besteht aus fünf Darstellungsbereichen. Diese Darstellungsbereiche sind untereinander in der folgenden Reihenfolge dargestellt und in der Abbildung 5.15 mit Nummern markiert.

1. Wissenschaftlicher Titel
2. Mehrsprachige Artenauswahl
3. Karte mit Legende
4. Ortssuche
5. Artinformationen

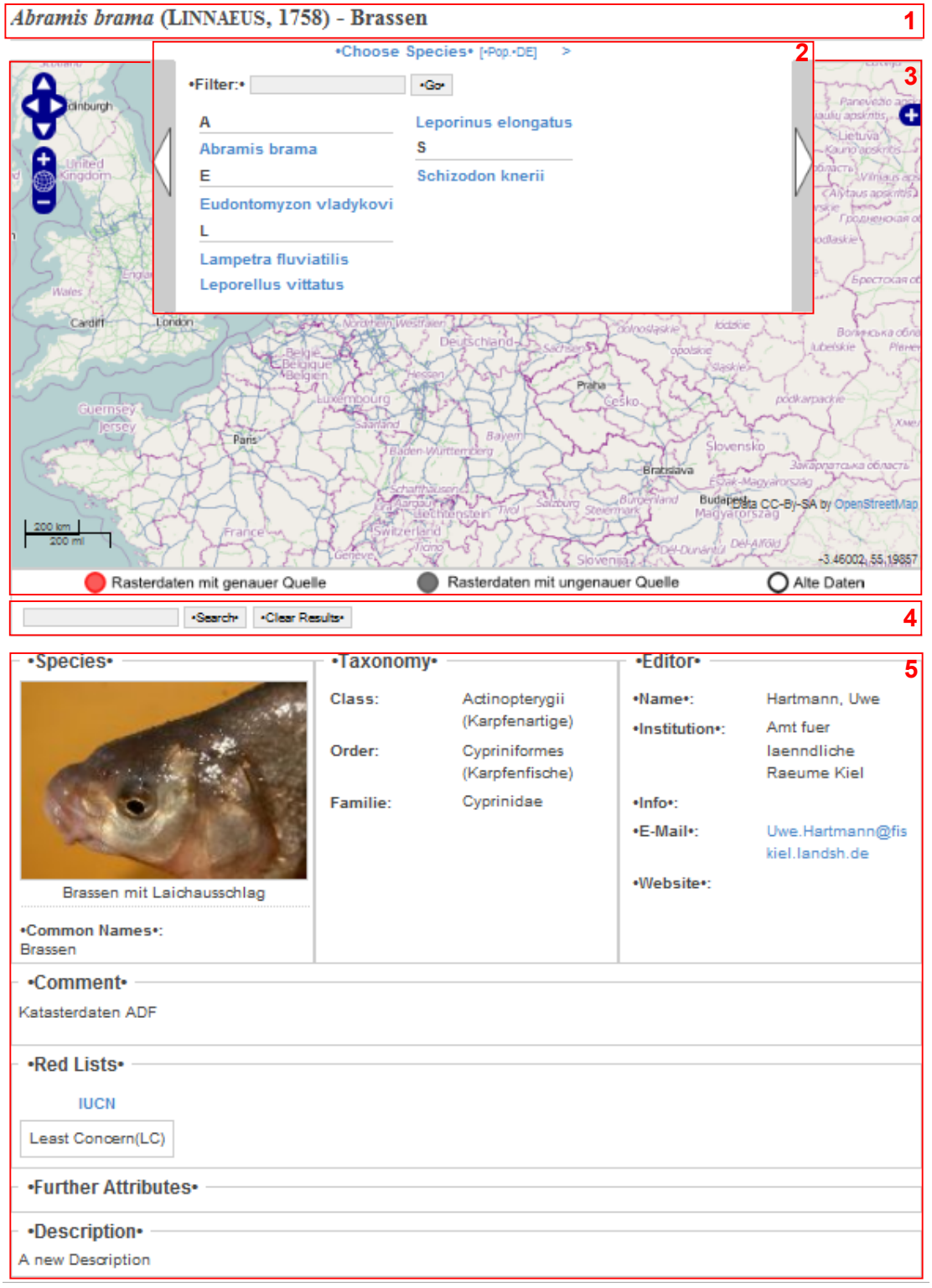

Abbildung 5.15: Frontend-Ansicht „Species“

Die View zeigt beim ersten Aufruf nur die Arten-Auswahl und eine Karte mit der Ortssuche an. Erst nach dem eine Art über die Arten-Auswahl gewählt wurde werden alle Teile der View dargestellt. Dies geschieht mit der Methode *show()* des Controllers. Dabei wird zu erst die View für das Aufgabenpaket Species initialisiert, dann das dazugehörige Model. Bevor in der View die *display()*-Methode aufgerufen wird, wird das Model der View übergeben. Das Model des Paketes besitzt in diesem Fall nur *get()*-Methoden, da die View nur zur Darstellung genutzt wird. Aus diesem Grund werden in diesem Model auch keine Tableobjekte angesprochen. Neben dem Parameter für den Task, wird zusätzlich eine eindeutige Identifikation einer Art übergeben. Anhand dieser Identifikation lädt das Model auf Anfrage der View die darzustellenden Daten. In der View werden diese Daten an ein *Template* übergeben.

Im *Template* setzt sich der als Überschrift genutzte, wissenschaftliche Titel aus einzelnen Feldern eines Eintrages aus der Tabelle „species“ zusammen. Darunter folgt die Artenauswahl.

Nach dem Aufruf der Seite kann eine Art über ihren wissenschaftlichen Namen ausgewählt werden. Alle Namen sind alphabetisch sortiert. Darüber hinaus ist es möglich, die Menge der dargestellten Namen über einen Filter zu reduzieren. Ist der wissenschaftliche Name nicht bekannt, kann die Anzeige der Auswahl auch auf die Darstellung der Populärnamen umgeschaltet werden. Auch diese Namen sind alphabetisch sortiert und durch einen Filter reduzierbar. Die Auswahl ist zur besseren Übersichtlichkeit auf eine maximale Größe begrenzt. Um trotzdem Zugriff auf alle Arten zu ermöglichen, gibt es Pfeile, die die Anzeige der Auswahl per Klick um je eine Spalte nach links oder rechts verschieben. Diese Verschiebung wird zur Verbesserung der Nutzerfreundlichkeit in einem *Cookie* gespeichert, so dass der Nutzer beim erneuten Öffnen der Auswahl an derselben Stelle wie zuvor weiter suchen kann. Dies ist unabhängig davon, ob die Seite neu geladen wurde. Da eine Art im ersten Teil des Namens immer die zugehörige Gruppe beinhaltet, ist die Option vorhanden, direkt zwischen den benachbarten Arten, die sich aus diesem Grund oft in derselben Gruppe befinden zu wechseln. Hierzu sind zwei kleine Pfeile in der Leiste, in der auch die Auswahl der Sprache möglich ist, dargestellt.

Die genannten Funktionen werden durch verschiedene, eigens dazu implementierte JavaScript-Methoden bereitgestellt. Auch wenn es so scheint, ist kein *AJAX* verwendet worden. Die Darstellung ist durch das einfache Ein- und Ausblenden von Objekten des *Document Object Models* realisiert worden. Einzig das Umschalten der Sprache, das heißt zum Beispiel von wissenschaftlichen Bezeichnungen der Arten zu deutschen Populärnamen, wird durch das erneute Laden der Seite realisiert. Dies geschieht, um nur die nötigen Namen zu erfassen und damit die Menge der Abfragen an die Datenbank zu verringern, um somit die *Performanz* beim Laden einer Seite für eine Art zu erhöhen. Da ein häufiges Umschalten der Sprachen seltener vorkommt als das Laden der Ansicht verschiedener Arten, wurde diese

Lösung gewählt.

Die Kartendarstellung ist, wie gefordert, so eingearbeitet, dass sie so groß wie möglich dargestellt wird, ohne dass dem Nutzer die darauf folgenden Inhalte verborgen bleiben und er sie deswegen übersieht. Die dargestellte Karte besitzt die für *GIS*-Anwendungen typische Navigation. Mit der Maus kann die Karte innerhalb der, durch den Parameter für die Eingrenzung der Darstellung, festgelegten Grenzen bewegt werden. Ergänzend ist das Vergrößern und Verkleinern der Karte mittels Mausrad möglich. Per Doppelklick kann die Karte um einen festen Faktor vergrößert werden. Weiter gibt es eingeblendete Schaltflächen, über die dieselben Effekte wie mit der Maus erzielt werden können.

Die in der Karte dargestellten Informationen kommen aus verschiedenen Quellen, dazu werden im Model die einzelnen Tabellen mit Bezug auf die Kartendarstellung ausgelesen und für die Verwendung in der View aufbereitet. Im Model wird hierfür zuerst der in den Parametern festgelegte Anzeigebereich aus der Tabelle „displayarea“ geladen. Im nächsten Schritt werden alle für die Karte zur Verfügung stehenden Ebenen aus der Tabelle „Layers“ angefordert. Die Anzahl der Ebenen setzt sich aus den Basisebenen, die keine Zuordnung zu einer Art besitzen, und den artspezifischen Ebenen, mit einer Zuordnung zur jeweils angeforderten Art, zusammen. Im letzten Schritt werden alle Vorkommen einer Art aus der Tabelle „location“ geladen, deren Verortung im vorher geladenen Geltungsbereich liegt.

Das Vorgehen beim Laden der Vorkommen ist abhängig vom Status eines Nutzers. Ist der Nutzer nicht angemeldet, werden, sofern ein Verfremdungsmechanismus existiert und für die View eingestellt ist, alle Daten in verfremdeter Form dargestellt. Ist dieser Mechanismus nicht in der View eingeschaltet, werden alle Daten an ihren realen Positionen dargestellt. Im Fall der verfremdeten Darstellung werden anstelle der Punkte einzelner Vorkommen, die Punkte der zu den Vorkommen passenden Verfremdungsbereiche geladen. Von den Vorkommen werden nur die für eine darzustellende Auflistung, in dem jeweiligen Verfremdungspunkt, benötigten Daten eines Vorkommens geladen. Alle weiteren Daten eines Datensatzes werden ignoriert. Dadurch kann die Menge der zu übertragenen Daten verringert werden.

Ist der Nutzer registriert, werden ihm alle Daten an den originalen Positionen dargestellt. Es gibt jedoch die Ausnahme, dass jeder Nutzer die genaue Anzeige der Daten, die er einträgt, verweigern kann. In diesem Fall werden die Daten, sofern ein Verfremdungsmechanismus installiert und eingestellt ist, auch registrierten Nutzern in verfremdeter Form dargestellt. Ist der Nutzer der Eigentümer der Daten oder gibt es keine Möglichkeit zur Verfremdung, werden auch diese Daten genau dargestellt. Um die Datenübersicht zu verbessern, werden die Daten im Model in die in den Anforderungen definierten Kategorien sortiert:

- Historisch
- Ungenau
- Genau

Daten sind, wie gefordert historisch, wenn das Datum der Entdeckung des Vorkommens vor dem Jahr 1980 liegt. Als ungenaue Daten gelten gemäß den Anforderungen alle Daten, die entweder nicht exakt datiert sind oder explizit als ungenau über das Attribut „locationquality“ in der Tabelle „location“ markiert wurden. Analog dazu sind genaue Daten als solche markiert und besitzen ein genaues Datum der Entdeckung des Vorkommens. Die daraus resultierende Sortierung wird in der Karte in Form von einzelnen Ebenen je Kategorie genutzt. Es ist dem Nutzer freigestellt, einzelne Kategorien auszublenden.

Die einzelnen Vorkommen werden in Form von gefüllten Kreisen dargestellt. Fährt man mit der Maus über einen Punkt, werden weitere Informationen zu dem Punkt in einem Popup dargestellt. Diese Informationen beziehen sich in der Gastansicht auf die Quelle und in der Ansicht für registrierte Nutzer auf das Entdeckungsdatum und den Entdecker. Um die Kategorien der Punkte zu erkennen, ist zudem jeder Kategorie eine Farbe zugeordnet. Abbildung 5.16 zeigt die Farben, die in einer Karte abgebildet werden können und ihre Bedeutungen.

Abbildung 5.16: Abgebildete Farben und Symbole in der Ansicht für Gäste

Ergänzend gibt es weitere Farben, wie sie in Abbildung 5.17 zu sehen sind, für die Ansicht der registrierten Nutzer. Diese Farben geben genaueren Aufschluss über die Daten. Daten aus dem Besitz eines Nutzers werden mit einer grünen Linie umrandet. Nicht zur genauen Darstellung freigegeben Daten werden mit einer grauen Linie umrandet.

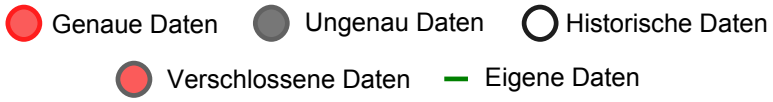

Abbildung 5.17: Abgebildete Farben und Symbole in der Ansicht für angemeldete Nutzer

Als Navigationshilfe ist unter der Kartendarstellung ein Suchfeld eingefügt. Mit diesem Suchfeld können über den bereits vorgestellten *OSM* Geocoding-Dienst Nominatim Orte gesucht werden. Gefundene Orte werden in Form von Links unter dem Suchfeld angezeigt. Der Nutzer kann über die Links zu den dazugehörigen

Orten in der Karte wechseln. Der Suchservice wurde mit JSON with Padding (JSONP) realisiert.

Über JSONP wird ein HTML-Script-Tag mittels JavaScript dynamisch erzeugt. Dieses Script-Tag beinhaltet den Aufruf einer fest definierten Methode mit den Daten der entfernten Quelle. Mit dieser Vorgehensweise wird die in den Browsern, im Bezug auf JavaScript, vorhandene „*Same Origin Policy*“ umgangen. Diese Richtlinie verbietet eine Nutzung anderer Datenquellen außerhalb der eigenen Domain, mit Hilfe der *AJAX* Technologie. Da es jedoch prinzipiell erlaubt ist, JavaScript-Dateien aus anderen Quellen einzubinden, kann diese Sperre mit Hilfe von JSONP umgangen werden, ohne dass ein installierter *Proxyserver* zur Umleitung von, durch *AJAX* verwendete, XMLHttpRequests auf dem Webserver vorhanden sein muss.

Das dynamische Erstellen von Script-Tags stellt jedoch ein Sicherheitsrisiko dar. Im Falle der Suche könnten über das Eingabefeld eigene Tags eingefügt werden, sodass zum Beispiel bösartiger Code in Form von JavaScript in das Formularfeld eingesetzt werden könnte. Nach dem die Suchfunktion durch den Button ausgelöst wurde, würde dieser Code, sofern er nicht vorher überprüft wurde, ebenfalls dynamisch eingebunden werden. Aus diesem Grund wird jede Eingabe sehr restriktiv gefiltert, indem nur Buchstaben und typische Satzzeichen erlaubt sind. In Abbildung 5.18 ist der dafür verantwortliche reguläre Ausdruck dargestellt.

```
var regexp = /^[a-zA-Z0-9 -'"`,\.]*$/;
```

Abbildung 5.18: Regulärer Ausdruck zur Filterung der Sucheingaben (JavaScript)

Die Artinformationen, ebenfalls in Abbildung 5.15 zu sehen, stellen den wissenschaftlichen Teil der Webseite dar. Jede Art wird mit einer Beschreibung, einem Bild und der Einordnung in die *Taxonomie* erklärt. Diese Darstellung wird um eine Zuordnung der Art in bekannte rote Listen und die Möglichkeit, weitere weniger wichtige Attribute anzuzeigen, ergänzt. Alle dargestellten Daten werden über *get()*-Methoden des Models aus der Datenbank in die Webseite eingefügt, sofern dazu Datensätze vorhanden sind.

OpenLayers Helper

Für die Kartendarstellung und dem damit verbundenen Suchfeld wurde eine eigene Hilfsklasse implementiert, die dafür zuständig ist den JavaScript-Code zu erzeugen, der für die Darstellung benötigt wird.

Mit Hilfe des Konstruktors werden der Klasse alle benötigten Daten für die Erzeugung eines Objektes übergeben. Das Objekt besitzt zwei öffentliche Methoden, die Methode *addMap()* dient zur Konstruktion der Kartendarstellung und

die Methode *addLocationSearch()* dient zur Konstruktion des Suchformulars. Die öffentliche Methode *addMap()* nutzt weitere private Methoden, mit denen unter anderem die JavaScript-Bibliotheken OpenLayers und Proj4js in den HTML-Head eingefügt werden. Außerdem wird ein Skript erzeugt, das eine Karte in einem Div-Container instanziiert und mit den angesprochenen Navigationselementen für die Karte ausstattet. Innerhalb dieses Skriptes werden der Karte zusätzlich alle durch das Model angefragten Ebenen angefügt. Dazu werden zu erst alle einer Ebene angehörenden Parameter passend sortiert. Wenn eine Ebene in einer von der Karte abweichenden *Projektion* vorliegt, wird zusätzlich eine PROJ.4-Definition zur Transformation mit Hilfe der Proj4js-Bibilothek, in das Skript eingefügt. Daraufhin wird eine Ebene zur Karte hinzugefügt.

Nachdem alle Ebenen in der Karte eingeordnet wurden, werden die Vorkommen eingetragen. Hierzu werden für die drei Kategorien Vektorebenen definiert. Jede Vektorebene wird mit einem Stilobjekt verknüpft, um die Darstellung der Vorkommen auf der Karte mit den bereits erklärten Formen und Farben zu konfigurieren. Das Stilobjekt enthält dafür Grundeinstellungen zur Darstellung eines Punktes, die mit Hilfe eines Regelwerks modifiziert werden können. Die Grundeinstellung für die Darstellung eines Vorkommens ist als rot eingefärbter Kreis mit rotem Rand definiert. Zuletzt wird jeder Punkt als Geometrie im Well Known Text Format instanziiert und mit einem Mouseover-Event sowie einem Popup verbunden. Um die Darstellung eines Punktes mit Hilfe des angesprochenen Regelwerks zu verändern, besitzt jeder Punkt ein Attribut „quality“. Der Wert dieses Attributes bestimmt die Darstellung des Punktes gemäß den definierten Regeln. Hat ein Punkt beispielsweise nur ungenaue Koordinaten, bekommt sein „quality“-Attribut den dafür definierten Wert und wird aufgrund des Regelwerkes modifiziert, woraufhin er mit einer grauen Füllung und einem grauen Rand dargestellt wird. Danach wird er der zugehörigen Vektorebene hinzugefügt. Das daraus entstandene Skript wird dem im Voraus beschriebenen Skript zur Erzeugung der Karte angefügt. Die *addMap()*-Methode gibt ein Div-Element zurück das per echo-Befehl in die Seite eingefügt wird. Dieses Div-Element beinhaltet die Karte und eine JavaScript-Methode, die nach dem Rendern des Div-Elements die Karte rendert.

Neben der *addMap()*-Methode besitzt die Hilfsklasse eine Methode *addLocationSearch()*, die ebenfalls öffentlich zugänglich ist. Sie erzeugt die nötigen JavaScript-Methoden, um ein Suchfeld mit einem Button zum Start der Suche und einem Button zum Löschen der Suchergebnisse nutzbar zu machen. Sie definiert Methoden zum Senden von Anfragen an den *OSM*-Service Nominatim und zur Aufbereitung einer Liste mit Hilfe von JSONP. Weiterhin werden Methoden definiert, die es ermöglichen über einen Klick auf ein Listenelement, die Karte an den, durch das gewählte Listenelement, vertretenen Punkt zu zentrieren. Das entstandene Skript wird ebenfalls in den HTML-Head eingefügt. Die Methode gibt ein Div-Element mit den angesprochenen Input-Elementen zurück, es kann über einen einfachen echo-Befehl mit PHP in die Seite eingefügt werden.

Locations View

Die Location View kann neben der Species View über den Menümanager ausgewählt werden. Die Location View besitzt im Vergleich zur Spieces View nur einen Parameter „Mail Tag“. Dieser Parameter ist Teil der beschriebenen, impliziten Anforderungen, dass eine Benachrichtigung in Folge eines neuen Eintrags in der Tabelle „location“, mit Hilfe der Location View, verschickt werden soll. Die Benachrichtigung vermeidet unnötige Überprüfungen auf neue Fundorte im Backend und sorgt für eine einfachere Handhabung der Überprüfungen. Der Parameter wird dazu verwendet, die versendeten Benachrichtigungen mit einem E-Mail Client filtern zu können. Da die View nur in einem gesonderten internen Bereich dargestellt werden soll, fehlt ihr augenscheinlich ein Parameter, der dies bestimmt. Hier kommt das Joomla!-*CMS* zum Einsatz. Bei der Auswahl im Menümanager kann der View ein Access Level hinzugefügt werden.

Es gibt insgesamt drei Access Level:

- **Public** – Alle Nutzer
- **Registered** – Nur angemeldete Nutzer
- **Special** – Nur angemeldete Nutzer in einer spezielleren Gruppe als „Registered“

Die Location View kann also mit Hilfe der Markierung des Access Level „Registered“ auf die, im Rahmen des Nutzerkonzeptes gewünschte, interne Zugangsberechtigung eingestellt werden.

Die in Abbildung 5.19 gezeigte Location View besitzt im Gegensatz zu Species View eine weniger vielschichtige Darstellung. Die View ist aus zwei Teilen zusammengesetzt, ein Teil ist ein Formular zur Erfassung von Vorkommen, der andere Teil eine Karte für die Vorschau eines Punktes. Jeder registrierte Nutzer kann, sofern im Menümanager das Access Level auf „registered“ eingestellt ist, nach der Authentifizierung Vorkommen erfassen.

Für die Erfassung wird im Formular in Pflichtfelder und optionale Angaben unterschieden. Die, durch die Klasse JHTML, in Joomla! bereitgestellte Behavior „Formvalidator“ unterstützt den Nutzer beim Ausfüllen, in dem das Senden des Formulars bei fehlerhaften Angaben mit Hilfe von JavaScript unterbrochen wird. Zusätzlich werden die fehlerhaften Angaben durch eine rote Einfärbung der Felder hervorgehoben. Für die Speicherung besitzt der Frontend Controller eine Methode *save()*. Anhand dieser Methode wird wie im Backend eine Aktionskette angestoßen, in der das Model in der Methode *store()* die übermittelten Daten aus dem Request aufbereitet und diese daraufhin an das Tableobjekt weiter gibt, welches für die Speicherung verantwortlich ist. Sollte der Nutzer das Formular trotz der JavaScript-gestützten Validierung abschicken können, wird der ungenügende Datensatz bei einer zweiten Überprüfung im Tableobjekt verworfen.

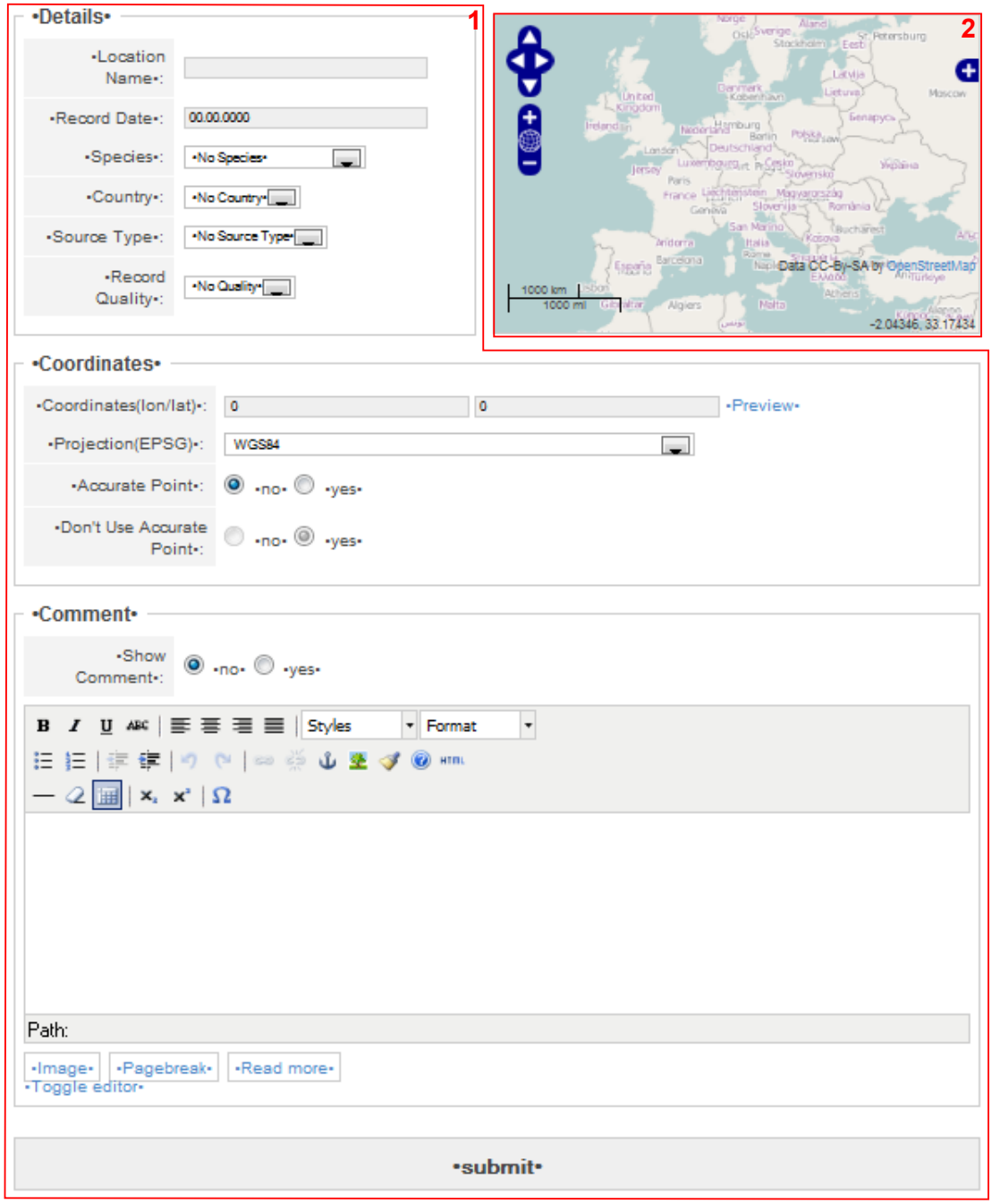

Abbildung 5.19: Frontend-Ansicht „Location“

Ist die Speicherung jedoch erfolgreich, wird über die private Methode *sendNotification()* eine E-Mail zur Benachrichtigung über das Joomla!-*CMS*, an die in der Joomla!-Grundkonfiguration angegebene E-Mail-Adresse versendet. In dieser Methode kommt der im Menümanager eingetragene Parameter zur Anwendung, er wird an den Anfang des Betreffs angefügt und ermöglicht die Filterung der versendeten E-Mails mit üblichen E-Mail-Clients.

Im Rahmen dieser View zeigt sich auch der Vorteil der in dieser Arbeit umgesetzten Einheitlichkeit zu dem Joomla!-Framework und dem *CMS* an sich. Die Location View und das damit verbundene Model sind aus technischer Sicht als analog mit gewissen Einschränkungen, zu dem Aufgabenpaket Location des Backends zu betrachten.

Vergleicht man die Location-Formularansicht des Frontends mit der Formularansicht des Backends, sind diese beiden Formulare sehr ähnlich. Im Frontend wurden lediglich die Felder mit den Titeln „Published“ und „Editor“ entfernt. Das Feld „Published“ wurde entfernt, da ein im Frontend eingefügtes Vorkommen, bis zur Begutachtung durch einen autoritären Nutzer im Backend, nicht zur Darstellung freigegeben ist. Das Editor Feld ist ebenfalls nicht nötig, da ein Bearbeiter eindeutig anhand seiner Identifikation im Joomla!-*CMS* einem Eintrag in der Tabelle „editor“ zugeordnet werden kann und es einem Nutzer nicht gestattet ist, im Namen eines anderen Nutzers Vorkommen zu erfassen.

Die Einbindung der Karte wird, wie im Backend, über eine JavaScript-Datei mit Methoden zum Anzeigen und Übernehmen von Koordinaten gelöst. Wie auch in den anderen Vorschaukarten im Backend wurde OpenLayers mit Karten von *OSM* für die Darstellung des örtlichen Bezugs eines Punktes festgelegt. Von der Verwendung des OpenLayers Helper wurde abgesehen, da diese Hilfsklasse speziell für die Darstellung komplizierterer Inhalte entworfen wurde und die Einbindung einer kleinen JavaScript-Datei in diesem Fall einfacher und ressourcenschonender möglich ist.

In dem Model sind bis auf eine Ausnahme identische Methoden zu finden. Dies trifft auf die *store()*-Methode zur Speicherung eines Datensatzes sowie auf die *get()*-Methoden zu. Einzig eine Methode zur Bestimmung des, zu einem Joomla!-Nutzer gehörenden, Eintrags in der „editor“ Tabelle aus der Datenbank ist abweichend hinzugefügt worden. Gleiches gilt für das, durch das Model zur Speicherung der Daten, genutzte Tableobjekt. Faktisch ist die größte Abweichung im Model und im Tableobjekt auf das Fehlen von Methoden zum Löschen und Überarbeiten vorhandener Datensätze zurückzuführen, da diese Methoden im Frontend nicht benötigt werden.

Betrachtet man diese Art der Umsetzung, stellt sich die Frage, ob anstatt der Kopie von Klassenbeschreibungen aus dem Backend und der darauffolgenden An-

passungen, eine effizientere Implementierung durch den Import der Klassen mittels PHP „require“ oder „import“ Befehl im Frontend, möglich gewesen wäre. Prinzipiell wäre es möglich, es wurde jedoch bewusst nicht so umgesetzt. Die Art der Umsetzung wurde gewählt, weil beide Komponenten, obwohl sie aufgrund der gewählten Namenskonventionen denselben Namen besitzen, eigentlich voneinander unabhängig betrieben werden können. Es ist also bei entsprechender Datenlage denkbar, die Frontend-Komponente durch die Art der Umsetzung auch ohne den Backend-Teil zu betreiben. Um diese Unabhängigkeit aufrecht zu erhalten, wurde, anstatt die Frontend-Komponente mit den Klassen der Backend-Komponente zu verknüpfen, die Lösung durch Kopien einzelner Klassen vorgezogen.

5.2.4 Plugin

Die Verfremdung der Vorkommen im Aufgabenpaket Locations ist ein wichtiges Anliegen um gefährdete Arten zu schützen. Da die Erfassung von Vorkommen im Backend sowie im Frontend stattfindet, musste eine Lösung gefunden werden, die eine Verfremdung in beiden Bereichen möglich macht. Zusätzlich bestand die Schwierigkeit, dass die Verfremdung vor allem bei einem länderübergreifenden Einsatz der Komponente unterschiedlich ausfallen kann. In Deutschland soll zum Beispiel, wie in der Anforderungsanalyse angesprochen, der Mittelpunkt eines *TK25*-Blattes genutzt werden. Um Ressourcen zu sparen, muss zusätzlich eine Grundlage für die persistente Ablage und Zuordnung zu einem solchen Punkt vorhanden sein, damit die Daten nicht bei jedem Aufruf erneut berechnet werden müssen. Wie im Bereich Datenbank erläutert, sind hierzu zwei Tabellen eingefügt worden. Die eine zur Speicherung von Verfremdungsdaten und die andere zu deren Verknüpfung mit den eingetragenen Vorkommen.

Für die Umsetzung in Joomla! fiel die Wahl auf die in Joomla! einsetzbaren Plugins. Plugins sind wie in der Analyse von Joomla! dargestellt im Frontend sowie im Backend ansprechbar. Zusätzlich werden sie in Gruppen organisiert. Der Vorteil liegt darin, dass Plugins nie direkt angesprochen werden, stattdessen werden in Joomla! immer die Plugingruppen angesprochen. Dazu definieren Plugins Events. Diese Events können von anderen Teilen des *CMS*, unter anderem auch Komponenten, mit Bezug auf eine Gruppe ausgelöst werden. In einem Bereich ausgelöste Events werden an die bei der Auslösung des Events angegebene Plugingruppe weitergegeben. Jedes Plugin dieser Gruppe mit einer übereinstimmenden Event-Definition reagiert auf das Event. Wie durch den Begriff Event angedeutet, funktionieren Plugins mit Hilfe einer in Joomla! umgesetzten Variante des Observer-Musters. Das Observer-Muster ist in Abbildung 5.20 zu sehen.

Durch die Unabhängigkeit der Umsetzung ist das Potential vorhanden, verschiedene Plugins für Verfremdungen zu definieren. Indem ein Plugin einer Region zugeordnet wird, kann je nach Region eine andere Verfremdung genutzt werden. Anhand des Plugins für Deutschland soll diese Funktion verdeutlicht werden. Das

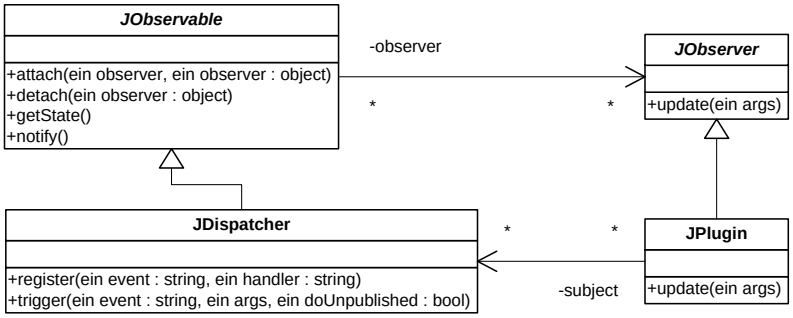

Abbildung 5.20: JPlugin im Observer-Muster

Plugin für *TK25*-Verfremdung in Deutschland definiert zwei Events das Event onSaveLocation und onRemoveLocation. Diese Events sind durch gleichnamige Methoden in der Plugin-Klasse repräsentiert. Um das Plugin zu verwenden, besitzt das für die Speicherung und Löschung von Vorkommen verantwortliche Tableobjekt, überladene *store()*- und *delete()*-Methoden. Nach erfolgreicher Ausführung der SQL-Operationen werden die angegebenen Events in diesen Methoden ausgelöst. Abbildung 5.21 zeigt den Code, der zur Auslösung eines Events benötigt wird.

```
JPluginHelper::importPlugin( 'biodiversity' );
$dispatcher =& JDispatcher::getInstance();
$results = $dispatcher->trigger( 'onSaveLocation', array(&$table));
```

Abbildung 5.21: Code zum Auslösen des Events „onSaveLocation“ für die Gruppe „biodiversity“

Nach dem die Events durch das Framework an die bei der Auslösung definierte Plugingruppe weitergeleitet worden sind, reagieren alle Plugins dieser Gruppe, sofern sie eine passende Event-Definition besitzen, auf das Event. In diesem Fall reagiert das Plugin für die deutsche *TK25*-Verfremdung und vergleicht das mit dem Event übermittelte Vorkommen mit den in der Datenbank vorhandenen Verfremdungsdaten. Verfremdungsdaten sind in Form eines *Polygons* für ihren Geltungsbereich beschrieben und besitzen einen, für diesen Bereich zu verwendenden, Mittelpunkt für die Verfremdung. Der Punkt des Vorkommens wird darauf überprüft, ob er in dem Bereich einer Verfremdung liegt. Ist dies der Fall, wird eine Verknüpfung des Vorkommens und der Verfremdung in der entsprechenden Tabelle eingetragen. Sollte eine Verknüpfung bereits vorhanden sein, wird diese Verknüpfung falls nötig aktualisiert. Bei dem Event onRemoveLocation, wird eine vorhandene Verknüpfung anhand der durch das Event gelieferten Identifikation aus der Verknüpfungstabelle entfernt.

Da jedem Verfremdungsdatensatz und jedem Plugin eine Länderidentifikation

zugeordnet werden kann, besteht die Option durch Kopieren und Ändern der Länderidentifikation ein weiteres Plugin für ein anderes Land zu erstellen. Neben der Bereitstellung eines Plugins müssen für die erfolgreiche Verknüpfung von Vorkommen und Verfremdungsbereichen, Verfremdungsdaten des Landes in die dafür vorgesehene Verfremdungstabelle importiert werden. Ist das geschehen, wird bei der Speicherung eines Vorkommens das neue Plugin ebenfalls angesprochen. Da eine fest vergebene Länderidentifikation, Teil eines Vorkommens sowie eines Plugins ist, wird nur ein Plugin tatsächlich angesprochen. Dies ist der Fall, da alle Plugins mit einer anderen Länderidentifikation als die des Vorkommens, die Bearbeitung des Datensatzes ablehnen. Das Plugin stellt somit eine Schnittstelle für die Verfremdung zur Verfügung, die über das Joomla!-Framework ansprechbar ist. Hierdurch entsteht die Möglichkeit verschiedene Verfremdungen in das System einzuführen, ohne die Anwendung in ihrem Kern zu verändern.

6 Test und Evaluation

Da die aus dieser Arbeit hervorgehende Anwendung eine prototypische Realisierung darstellt, wurden nur wenige Tests definiert. Es ist ein Test zum Verifizieren der Ergebnisse von Funktionen eingesetzt worden. Des Weiteren ist ein Test definiert, der den Ablauf bei der Erfassung eines Vorkommens simuliert. Größere Tests sollten im Rahmen einer Untersuchung der Einsatzfähigkeit durch das Projektteam erfolgen.

6.1 Tests

Für alle Test wird von einem Startzustand ausgegangen. Dieser Startzustand simuliert eine intakte Anwendung mit validen Datensätzen in der durch die Anwendung verwendeten Datenbank. Dazu sind drei Arten definiert, zu denen jeweils ein bis zwei Vorkommen eingetragen sind. Ein Bearbeiter ist definiert, der den Arten zugeordnet ist. Zusätzlich sind zwei Kartenebenen sowie ein Bereich für die Sichtgrenzen und ein Land für die Zuordnung der Karte definiert. Es gibt eine *Taxonomie* mit zwei Ästen unterhalb der Wurzel. Jeder dieser Äste besitzt drei weitere Stufen mit je einem Eintrag. Für diese Stufen gibt es drei Hierarchiebezeichnungen zur Auswahl. Eine Spezialität ist zur weiteren Gruppierung der Arten definiert. Für die roten Listen ist die Liste der IUCN[1] eingetragen.

6.1.1 Entwicklertest

Der Test dient zur Verifizierung einzelner Funktionen. Hierzu wird der vorausgehend definierte Startzustand verwendet. Der Startzustand wird vor jedem Funktionstest gesichert, daraufhin wird der Code für eine neue Funktion in die Anwendung eingeführt. Die neue Funktion wird ausgelöst. Im letzten Schritt wird das Resultat der Funktion durch den Vergleich mit dem gesicherten Aufbau festgehalten. Dieser Test wird mehrfach wiederholt. Entspricht das Ergebnis durchgehend den Erwartungen des Entwicklers, wird die getestete Funktion als vollständig betrachtet und fest in die Anwendung integriert.

6.1.2 Nutzertest

Mit diesem Test wird ein Teil der Datenmanagementfähigkeiten des Backends getestet. Dazu ist ein Nutzer, der später auch im Backend arbeitet, heranzuziehen.

[1] http://www.iucnredlist.org/

Der Nutzer ist ausgehend von dem bereits vorgestellten Startzustand dazu aufgefordert, ein Vorkommen mit Hilfe des Formulars im Backend zu erfassen und zu speichern.

6.2 Evaluation

Es wurden während der Entwicklung kleinere Tests durchgeführt, um Funktionen zu verifizieren. Des Weiteren wurde ein Test durchgeführt, der den Ablauf bei der Erfassung eines Vorkommens simuliert. Im Folgenden sind die Ergebnisse der Tests kurz dargestellt.

6.2.1 Entwicklertest

Im Rahmen des Entwicklertests wurde versucht, die Funktionsfähigkeit der Anwendung sicherzustellen. Es sind daher nur Funktionen in der Umsetzung vorhanden, die, die durch den Entwickler gewünschten Resultate erzielt haben. Da dieser Test inkrementell durchgeführt wurde und Einfluss auf die durch die Anwendung genutzten Funktionen hatte, ist die aus dieser Arbeit resultierende Anwendung die Summe aller positiv ausgefallenen Entwicklertests.

6.2.2 Nutzertest

Im Rahmen des Nutzertests fiel, auf das jedem Vorkommen zwar eine Person zugeordnet werden konnte, eine Quelle der Funddaten damit jedoch nicht eindeutig bestimmbar war. Dieses Ergebnis ist auf folgende Situation zurückzuführen. Trägt ein Nutzer zum Beispiel Daten aus der Literatur ein, wird seine Person als Auswerter der Literatur auch dem Vorkommen zugeordnet. Die Quelle des Vorkommens ist er jedoch nicht. Die Quelle ist, soweit es nicht anders erwähnt ist, das Buch selbst und der dazugehörige Autor. Diese Problematik ist vorher nicht beachtet worden, muss aber zukünftig berücksichtigt werden.

7 Bewertung und Ausblick

Ziel dieser Arbeit war die Entwicklung einer Komponente, die die aus den vorhergehenden Projekten zu diesem Thema entwickelten Anwendungsteile zu einer Joomla!-Anwendung zusammenfügt und um fehlende Kernfunktionen erweitert. Dabei sollte eine möglichst generische Komponente entstehen, die zum einen für verschieden Aufgabengebiete eingesetzt werden kann und zum anderen mit geringem Aufwand in Verbindung mit einer anderen Datenhaltung eingesetzt werden kann. Im Rahmen der Arbeit sind zwei Komponenten entstanden, eine für das Backend und eine für das Frontend des Joomla!-*CMS*. Ein exemplarischer Plugin zur Verfremdung von Geodaten ist ebenfalls realisiert worden. Die Backend-Komponente ermöglicht die Verwaltung aller Daten aus der Datenbank, basierend auf dem in der Realisierung vorgestellten Datenmodell. Die bereits vorhandene Artenverwaltung wurde neu implementiert und erweitert. Zusätzlich wurde eine komfortable Verwaltung von Vorschaubildern für eine Art hinzugefügt.

Die Verwaltung der Vorkommen ist mit einer Kartenvorschau umgesetzt worden, die eine Übernahme von Koordinaten aus der Karte genauso unterstützt wie das Eintragen und Transformieren eigener Koordinaten in dezimaler Form. Die Verwaltung für die Bearbeiter wurde weitgehend übernommen und mit zusätzlichen Formularfeldern versehen. Für die Verwaltung von roten Listen ist eine dynamischere Struktur eingeführt worden, die die Definition einer roten Liste vereinfachen soll.

Um zukünftig auch Informationen über die Einordnung einer Art in der biologischen *Taxonomie* erfassen zu können, wurde eine dynamische Verwaltung mit intuitiven Bedienelementen, implementiert. Mit dieser Verwaltung wurde eine einfache Struktur geschaffen, mit der, die in der Datenbank zur Abbildung der *Taxonomie* genutzten, *Nested Sets* verwaltet werden können, ohne mit der damit verbundenen Komplexität konfrontiert zu werden. Um die genauere Bestimmung der Daten zu ermöglichen, sind zusätzliche Informationen über eine allgemeine Verwaltung erfassbar und nutzbar. Für die Kartendarstellung wurde eine Verwaltung für Kartenebenen implementiert, die auf Basis der JavaScript-Bibliothek OpenLayers arbeitet. In diesem Zusammenhang ist eine Verwaltung für darstellbare Bereiche eingesetzt worden. Mit deren Hilfe können Gebiete und Abfragen begrenzt werden, um eine erhöhte *Performanz* in der Darstellung zu erreichen und einen Fokus setzen zu können.

Im Frontend ist eine mit Fachinformationen verbundene Darstellung entwickelt

worden, die mit verschiedenen Kartenebenen und Datenqualitäten genutzt werden kann. Mit Hilfe von OpenLayers sind alle Kartendarstellungen interaktiv nutzbar. Zur einfacheren Erzeugung der Karten wurde eine Helferklasse eingefügt, die durch ihren generischen Ansatz eine Karte mit diversen von OpenLayers unterstützten Ebenen zusammenstellen und zusätzlich Daten im Well Known Text Format aus einer geographischen Datenbank zu einzelnen Ebenen verbinden kann.

Um die Nutzbarkeit zu verbessern, wurde ein neues Menü zur Auswahl der Arten geschaffen. Dabei wurde die Auswahl der Sprachen übernommen und auf noch mehr Sprachen erweitert. Dieses Menü ist nicht mehr an die Position der Hauptnavigation gebunden. Es erleichtert dadurch die Orientierung des Nutzers, da das Hauptmenü nicht mehr verdeckt wird. Ergänzt wird das neue Menü für die Arten durch einen Filter zur Eingrenzung der angezeigten Arten.

Zur besseren Navigation in der Karte wurde der *OSM*-Service Nominatim eingeführt, mit dessen Hilfe Orte gesucht und in der Karte lokalisiert werden können. Diese Suche wird ebenfalls durch die Helferklasse zur Verfügung gestellt.

Im Frontend ist ein Formular erstellt worden, dass es registrierten Nutzern ermöglicht eigene Vorkommen einzutragen. Dieses Formular besitzt, wie im Backend, ebenfalls eine Vorschaukarte und die Möglichkeit Koordinaten aus der Karte zu übernehmen.

Im Sinne des Naturschutzes ist zusätzlich die Möglichkeit geschaffen worden, mit Hilfe von Plugins, diverse Verfremdung an den Geodaten dieser Anwendung durchzuführen, um nur den registrierten und damit bekannten Personen einen genaueren Blick auf die eingetragenen Geodaten zu gewähren, jedem Gast aber zumindest eine Vorstellung verschaffen zu können.

Mit dieser Arbeit ist die vorher in einzelnen, voneinander unabhängigen Teilen existierende Anwendung zu einer soliden Basisanwendung zusammengefasst worden, die es ermöglicht aufgrund der Joomla!-homogenen Entwicklung mit geringem Aufwand weitere Funktionalitäten hinzuzufügen.

Durch die Verwendung des Joomla!-Frameworks und des *MVC-Musters* ist die Abhängigkeit zur Datenhaltung so gering wie möglich realisiert worden. Für die Abfrage von Daten muss nur ein passendes Modelobjekt verändert werden. Für das Speichern der Daten kann ein Tableobjekt angepasst werden. Dabei wurde mit eindeutigen im Framework definierten Methoden gearbeitet, um weitere Abhängigkeiten auszuschließen. Zusätzlich wurde Gebrauch von den in Joomla! üblichen Konventionen und Entwicklungsmustern gemacht.

Einige Dinge konnten oder sollten im Rahmen dieser Arbeit noch nicht realisiert werden. Im Folgenden werden weitere Punkte beschrieben, deren Realisierung zu-

künftig von Vorteil wäre.

Mehrsprachigkeit wurde in der Form, dass alle in der Datenbank befindlichen Tabellen in mehreren Sprachen vorliegen, noch nicht realisiert. Dies sollte allerdings in Zukunft in Betracht gezogen werden. Aus Sicht des Autors dieser Arbeit empfiehlt es sich nicht, eine eigene Umsetzung zu implementieren. Es ist sinnvoller die Joomla!-Erweiterung Joom!Fish als eine der bekanntesten Erweiterungen für die Übersetzung von Inhalten im Joomla!-*CMS* auf ihre Anwendbarkeit zu überprüfen. Mit Hilfe von Joom!Fish können üblicherweise Datenbankeinträge mit einer Joom!Fish Übersetzung verknüpft werden, ohne weitere Datenbanktabellen zu erstellen. Dazu müssen die Datenbanktabellen mit Hilfe von XML erfasst werden. Diese Erweiterung bietet den Vorteil, dass das Datenbankmodell, nicht nur aufgrund von Übersetzungen, komplizierter wird.

Die Realisierung von Abfragefunktionen im Bezug auf Mengen und Flächen als typische *GIS*-Anwendung wurde aufgrund der fehlenden Möglichkeiten mit MySQL nicht umgesetzt. Sollte in der Zukunft eine andere Datenhaltung in Form von PostgreSQL oder einem *Mapserver* zur Verfügung stehen, kann diese Funktionalität jedoch einfach durch ein zusätzliches Modul eingearbeitet werden. Im Verlauf der Arbeit fand zum Beispiel der Einsatz eines *Mapservers* erwähnung. Mit dessen Hilfe könnte ein Modul realisiert werden, das in Verbindung mit dem *Mapserver* und den vorhandenen Daten eine solche Abfrage realisiert. In diesem Zusammenhang wäre eine Erweiterung der derzeitigen Komponenten auf eine Datenhaltung, wie sie im Bereich der Lösungsansätze in Verbindung mit einem Geoserver beschrieben wurde, ebenfalls denkbar.

Ein letzter Punkt ist die Einbindung von OpenLayers. Zukünftig wäre eine eigenständige OpenLayers Verwaltung denkbar, die es ermöglicht diese Bibliothek, auf ähnliche Weise wie es bei den Texteditoren in Joomla! der Fall ist, einzubinden. Es wäre auch für die Community des *CMS* erstrebenswert, Joomla! als *GIS*-fähig anbieten zu können. Als erster Ansatz kann die realisierte Helferklasse angesehen werden, sie ist jedoch noch nicht flexibel genug, um in jedem Kontext einsetzbar zu sein. Eine Zugriffsmöglichkeit auf diese Klasse durch verschiedene Komponenten, wie es mit Hilfe eines Plugins bei den Texteditoren möglich ist, ist ebenfalls noch nicht vorhanden. Es ist jedoch möglich OpenLayers in einem Plugin, mit einer zu den Texteditoren vergleichbaren Struktur zu nutzen. Ein erster Ansatz entstand während dieser Arbeit, ist jedoch im Verlauf der Arbeit aufgrund des unnötigen Aufwands im Vergleich zu den benötigten Funktionen wieder verworfen worden.

Glossar

AJAX
siehe *Asynchronous JavaScript and XML*

Asynchronous JavaScript and XML
Methode Daten zwischen Server und Client auszutauschen um einzelne Teile einer Webseite zu laden.
http://www.w3schools.com/ajax/default.asp

Biodiversität
Biologische Vielfalt, der davon abgeleitete Begriff Biodiversitäten wird in dieser Arbeit für die Bezeichnung von Teilen dieser Vielfalt verwendet.

Browser-Cookie
Möglichkeit Informationen bei einem Client (Browser) zu hinterlegen.
http://de.selfhtml.org/javascript/objekte/document.htm#cookie

CMS
siehe *Content Management System*

Content Management System
CMS sind darauf ausgelegt darzustellende Informationen verwalten zu können und im besten Fall von der Art der Darstellung zu trennen.
http://www.joomla.de/

Cross-Site-Scripting
Methode zum Einschleusen von meist Browser-basierten Scripten in eine sonst glaubwürdige Webseite.
http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

Document Object Model
Plattform unabhängige Schnittstelle für dynamische Veränderungen an der Struktur, dem Inhalt und dem Still eines Dokuments.
http://www.w3.org/DOM/#what

DOM
siehe *Document Object Model*

Ellipsoid
Dreidimensionales Gegenstück zur zweidimensionalen elliptischen Form.

Entity Relationship Diagramm
Eine grafische Darstellung der Tabellenstruktur in einer Datenbank.
http://www.itwissen.info/definition/lexikon/ERD-entity-relationship-diagram-Entitaeten-Relationen-Diagramm.html

EPSG Code
Eindeutige Bezeichnung eines von der European Petroleum Survey Group katalogisierten Koordinatensystems.
http://www.epsg.org/geodetic.html

ERD
siehe *Entity Relationship Diagramm*

Extensible Stylesheet Language
Sprache zur Formatierung von XML-Daten.
http://de.selfhtml.org/xml/darstellung/xslgrundlagen.htm

Extensible Stylesheet Language Transformations
Sprache zur Formatierung von XML-Daten mit zusätzlichen Transformationsoptionen, um XML in andere Formate umzuwandeln.
http://de.selfhtml.org/xml/darstellung/xslgrundlagen.htm

Fallback-Controller
Controller-Klasse innerhalb einer Joomla!-Komponente, die aufgerufen wird, wenn das System einen angegebenen Controller nicht finden kann.

Geographic Information System
System zur Verwaltung, Erfassung und Analyse von geographischen Daten.
http://www.giswiki.org/wiki/Geographic_information_system

Geography Markup Language
XML-basierte Sprache zur Beschreibung von geographischen Merkmalen
http://www.opengeospatial.org/standards/gml

GIS
siehe *Geographic Information System*

GML
siehe *Geography Markup Language*

Graphical User Interface
GUI ist eine grafische Benutzeroberfläche einer Software, die dem Benutzer Interaktionen mit Hilfe grafischer Objekte erlaubt.

GUI
siehe *Graphical User Interface*

iFrame
In ein HTML-Dokument eingebetetter Frame zur unabhängigen Einbindung fremder Quellen in eine Webseite.
http://de.selfhtml.org/html/frames/eingebettete.htm

Koordinatenreferenzsystem
Koordinatensystem, das durch die Verknüpfung mit einem Datum auf die reale Welt bezogen ist.
http://www.en.giswiki.org/wiki/Koordinatenreferenzsystem

LAMP
Server mit dem Betriebssystem Linux, einem Apache-Webserver, einer MySQL-Datenbank und Unterstützung von PHP.

Mapserver
Server der OGC OpenGIS Web Services zur Verfügung stellt.

MBR
siehe *Minimum Bounding Rectangle*

Minimum Bounding Rectangle
Das kleinste eine Geometrie umschließende Rechteck.
http://dev.mysql.com/doc/refman/5.1/de/relations-on-geometry-mbr.html

Modal-Behavior
Durch das Joomla!-Framework automatisch erzeugtes JavaScript zur Darstellung von modalen Dialogen.

Model-View-Controller-Muster
Interaktionsmuster in der Präsentationsschicht von Software: Eine Eingabe wird durch den Controller empfangen und weiter vermittelt. Das Model verwaltet den Zustand der Software und verändert ihn, abhängig von der im Controller empfangenen Eingabe. Die View verwaltet die Präsentation und benötigt die darzustellenden Daten eines Zustands vom Model.[LR06]

MVC-Muster
siehe *Model-View-Controller-Muster*

Nested Sets
Methode Bäume mit nur einer Tabelle in SQL-Datenbanken darzustellen.
http://www.klempert.de/nested_sets/

OGC
siehe *Open Geospatial Consortium*

Open Geospatial Consortium
Non-Profit Organisation für die Standardisierung von ort- und raumbasierten Diensten.
http://www.opengeospatial.org/

Open Street Map
Freie Wiki-Weltkarte
http://www.openstreetmap.de/

OpenGIS Web Services
Diverse Spezifikationen für Webservices mit GIS-Bezug.
http://www.opengeospatial.org/standards

OSM
siehe *Open Street Map*

OWS
siehe *OpenGIS Web Services*

Performanz
Leistung

Persistenz
Datenspeicher zur permanenten Ablage von Daten.
http://www.itwissen.info/definition/lexikon/Persistenz-persistence.html

Polygon
Beliebige zweidimensionale geometrische Form auch Vieleck genannt.

Projektion
Methoden der Geographie um gekrümmte Oberflächen auf eine zweidimensionale Oberfläche zu übertragen.
http://www.giswiki.org/wiki/Projektion

Proxy
Software, die eine Anfrage stellvertretend für einen Client an einen Server stellt. Eine Anwendung zur Vermittlung von Kommunikation
http://www.squid-handbuch.de/hb/

Rasterdaten
Bilddaten aus Rasterpunkten zum Beispiel Pixeln. Ein Bild entsteht aus einer Anordnung von Rasterpunkten.
http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=1434

root
Administrativer Nutzer höchster Stufe auf einem Server

Same Origin Policy
Richtlinie eines Browsers, die sicherstellt, dass ein Script nur auf Dokumente aus der eigenen Quelle zugreifen kann.
http://www.w3.org/Security/wiki/Same_Origin_Policy

Simple Features Specification for SQL
Standard der eine GIS-kompatible Datenbank beschreibt.
http://www.opengeospatial.org/standards/sfs

SQL-Injection
Methode zum Einschleusen von SQL-Befehlen in eine Datenbank.
http://www.owasp.org/index.php/SQL_Injection

Taxonomie
Eine systematische Einordnung von Werten. Im Fall dieser Arbeit geht es um die Einordnung von biologischen Arten in ein Bestimmungssystem.

Template
Vorlage mit Platzhaltern für dynamische Inhalte.

Tile-Service
Service für den Bezug gekachelter Bilddaten.

TK25
siehe *Topographische Karte 1:25000*

TK25-Mittelpunkt
Mittelpunkt eines TK25-Quadranten.

Topographische Karte 1:25000
Detaillierte Kartendarstellung im Maßstab 1:25000 mit in Quadranten eingeteilten Flächen
http://www.lgn.niedersachsen.de/live/live.php?navigation_id=11068&article_id=51676&_psm

Tree-Behavior
Durch das Joomla!-Framework automatisch erzeugtes JavaScript zur Darstellung von baumähnlichen Hierarchien.

User Generated Content
Durch Nutzer einer Webseite erzeugte Inhalte.

Userinterface
Eine Schnittstelle die es dem Nutzer ermöglicht mit einer Software zu interagieren.

Vektordaten
Informationen die Mithilfe von Punkten, Linien und Flächen beschrieben werden. Für die Beschreibung der Lage dieser Elemente werden Koordinaten genutzt.
http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=1736

WCMS
siehe *Web Content Management System*

WCS
siehe *Web Coverage Service*

Web Content Management System
Content Management System mit Ausrichtung auf Websites

Web Coverage Service
Durch das OGC spezifizierter Webservice zur Lieferung von Rasterdaten in Form von Werten.
http://www.opengeospatial.org/standards/wcs

Web Feature Service
Durch das OGC spezifizierter Webservice zur Lieferung vektorbasierter Kartenansichten in einem XML-Format.
http://www.opengeospatial.org/standards/wfs

Web Feature Service – Transactional
Web Feature Service mit der Erweiterung, neben der Auslieferung von Daten auch vektorbasierte Daten einzufügen.

Web Map Service
Durch das OGC spezifizierter Webservice zur Lieferung von gerenderten Kartenbildern.
http://www.opengeospatial.org/standards/wms

Webservice
HTTP-basierter Service zur Lieferung von Daten
http://www.w3schools.com/webservices/ws_intro.asp

WFS
siehe *Web Feature Service*

WFS-T
siehe *Web Feature Service – Transactional*

WGS84
siehe *World Geodatic System 1984*

WMS
siehe *Web Map Service*

World Geodatic System 1984
Globales dreidimensionales Koordinatenreferenzsystem Es stellt eine weltweit einheitliche Grundlage für Positionsangaben dar.
http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html

XSL
siehe *Extensible Stylesheet Language*

XSLT
siehe *Extensible Stylesheet Language Transformations*

Abbildungsverzeichnis

2.1 Nutzermodell auf Basis des Vertrauens 6

3.1 Joomla!-Architektur 17
3.2 MVC-Muster 18
3.3 Ungenauigkeit des Minimum Bounding Rectangles 24
3.4 Ungenenaue Transformation durch eine Verschiebung des Mittelpunktes 28
3.5 Architektur mit Webclient und GeoServer als Mapserver 37

5.1 3-Tier-Client-Server-Architektur 45
5.2 Entity Relationship Diagramm der verwendeten Datenstruktur . . . 47
5.3 Struktur der Nested Sets 49
5.4 Für die Komponente verwendete Joomla!-Nutzertypen in Bezug auf ihre Rechte 51
5.5 Sitemap der Komponente im Backend 53
5.6 Klassendiagramm des Aufgabenpaketes Areas, exemplarisch für den Aufbau der anderen Pakete 55
5.7 Sequenzdiagramm für die Speicherung von Formulardaten am Beispiel des Aufgabenpaketes Areas 58
5.8 Verwaltung für die Taxonomie im Backend 60
5.9 Sequenzdiagramm der Move-Funktion 62
5.10 Bilderverwaltung mit Seitenkopf 65
5.11 Bilderverwaltung mit Parameter ohne Seitenkopf 65
5.12 Zuweisung eines Vorschaubildes zu einer Art aus Sicht eines Bearbeiters mit Fischabbildungen von Prof. Dr. Heiko Brunken 66
5.13 Sitemap der Komponente im Frontend 68
5.14 URL zum Aufruf der Komponente im Frontend 70
5.15 Frontend-Ansicht „Species“ 72
5.16 Abgebildete Farben und Symbole in der Ansicht für Gäste 75
5.17 Abgebildete Farben und Symbole in der Ansicht für angemeldete Nutzer 75
5.18 Regulärer Ausdruck zur Filterung der Sucheingaben (JavaScript) . . 76
5.19 Frontend-Ansicht „Location“ 79
5.20 JPlugin im Observer-Muster 82
5.21 Code zum Auslösen des Events „onSaveLocation“ für die Gruppe „biodiversity“ 82

Tabellenverzeichnis

3.1 Joomla! Anwendungselemente 18
3.2 Joomla! Nutzergruppen 19

Literaturverzeichnis

[Ada14] Adair, Mike: *Proj4js*. 2011-01-05 07:14:14. – http://trac.osgeo.org/proj4js/

[Ale09] Alex Kempkens: *Das Joomla!-Entwicklerhandbuch: Joomla!-Komponenten und -Templates programmieren mit dem Joomla!-Framework*. München [u.a.] : Addison-Wesley, 2009 (Open source library). – 542 S.. 1 CD-ROM (12 cm) : zahlr. Ill. und graph. Darst. – ISBN 978–3–8273–2323–1

[Anj10] Anja Ebersbach: *Joomla! 1.5: das umfassende Handbuch ; [Installation, Migration von 1.0, Anpassung, Erweiterung ; eigenes Design mit CSS und YAML, Extensions ; Webshop, Suchmaschinen-Optimierung, Sicherheit u.v.m.]*. 2., aktualisierte und stark erw. Aufl., 5., korrigierter Nachdr. Bonn : Galileo Press, 2010 (Galileo Computing). – 820 S.. 1 DVD-ROM (12 cm) : zahlr. Ill. – ISBN 978–3–89842–881–1

[BAP05] Windows Live (Hrsg.): *Bing Maps Developer Resources: Map SDKs, APIs, Tips, Training*. 2011-01-05 12:30:05. – http://www.microsoft.com/maps/developers/web.aspx

[BB17] Boenigk, Cornelia; Burger, Ralf ; Boenigk, Cornelia (Hrsg.); Burger, Ralf (Hrsg.): *PostgreSQL im Schnelldurchgang*. 2011-01-03 13:37:17. – http://www.postgresql.de/postgresql_outline.html

[GAP19] Google Inc. (Hrsg.): *Google Maps JavaScript API V3 – Allgemeines - Google Maps JavaScript API V3 - Google Code*. 2011-01-05 11:57:19. – http://code.google.com/intl/de-DE/apis/maps/documentation/javascript/basics.html

[GEX05] GeoExt Community (Hrsg.): *JavaScript Toolkit for Rich Web Mapping Applications — GeoExt v1.0*. 2011-01-02 18:40:05. – http://www.geoext.org/

[HSB19] Hochschule Bremen (Hrsg.): *Hochschule Bremen - Digitaler Fischartenatlas von Deutschland und Österreich*. 2010-12-28 17:49:19. – http://www.hs-bremen.de/internet/de/forschung/projekte/detail/index_18670.html

[JOO10] Joomla! Deutschland (Hrsg.); Joomla! Österreich (Hrsg.): *Joomla! Die Funktionen.* 2011-01-02 15:13:10. – http://www.joomla.de/die-funktionen.html

[Kar11] Karlsson, Anders ; Oracle (Hrsg.): *MySQL :: GIS and Spatial Extensions with MySQL.* 2011-01-03 10:04:11. – http://dev.mysql.com/tech-resources/articles/4.1/gis-with-mysql.html

[LR06] Lahres, Bernhard; Rayman, Gregor: *Praxisbuch Objektorientierung: von den Grundlagen zur Umsetzung.* 1. Aufl. Bonn : Galileo Press, 2006 (Galileo Computing). – 609 S. ; 25 cm : Ill., graph. Darst. – ISBN 978–3–89842–624–4

[Mar10] Marc Jansen: *OpenLayers: Webentwicklung mit dynamischen Karten und Geodaten.* München : Open Source Press, 2010. – 344 S. ; 176x240 mm : graph. Darst., Kt. – ISBN 978–3–937514–92–5

[MFB46] McKenna, Jeff; Fawcett, David; Butler, Howard ; University of Minnesota (Hrsg.): *An Introduction to MapServer — MapServer 5.6.5 documentation.* 2011-01-05 05:30:46. – http://mapserver.org/introduction.html#mapserver-overview

[MYS08] Oracle (Hrsg.): *MySQL :: MySQL Community Edition.* 2011-01-03 10:04:08. – http://www.mysql.de/products/community/

[MYS11] Oracle (Hrsg.): *MySQL :: MySQL 5.1 Referenzhandbuch :: 18.1 Einführung in die raumbezogenen Funktionen von MySQL.* 2011-01-03 10:04:11. – http://dev.mysql.com/doc/refman/5.1/de/gis-introduction.html

[MYS23] MySQLForge (Hrsg.): *GIS Functions - MySQL Forge Wiki.* 2011-01-03 10:04:23. – http://forge.mysql.com/wiki/GIS_Functions

[MYS47] Oracle (Hrsg.): *MySQL :: MySQL 5.1 Referenzhandbuch :: 1.9 Wie kompatibel zum SQL-Standard ist MySQL?* 2011-01-03 11:10:47. – http://dev.mysql.com/doc/refman/5.1/de/compatibility.html

[OLD10] OpenLayers Contributors (Hrsg.): *OpenLayers: Documentation.* 2011-01-02 15:13:10. – http://docs.openlayers.org/

[OSM38] Open Street Map Foundation (Hrsg.): *Main Page - OpenStreetMap Wiki.* 2011-01-05 13:51:38. – http://wiki.openstreetmap.org/wiki/Main_Page

[PGS35] Refractions Research (Hrsg.): *Refractions Research : PostGIS Technical Overview.* 2011-01-05 04:30:35. – http://www.refractions.net/products/postgis/

[PGS41] Refractions Research (Hrsg.): *PostGIS 1.5.2 Manual*. 2011-01-05 04:34:41. – http://postgis.refractions.net/documentation/manual-1.5/

[Pu43] Pumphrey, Mike; [u.a.]: *What is GeoServer - GeoServer*. 2011-01-05 05:30:43. – http://geoserver.org/display/GEOS/What+is+Geoserver

[RT10] Ramm, Frederik; Topf, Jochen: *OpenStreetMap: die freie Weltkarte nutzen und mitgestalten*. 3., überarb. und erw. Aufl. Berlin : Lehmanns Media, 2010. – XIV, 337 S. ; 240 mm x 170 mm : Ill., graph. Darst., Kt.. – ISBN 978–3–86541–375–8

[SMY24] Monty Program Ab (Hrsg.): *Save MySQL! THANKS FOR ALL THE HELP!* 2011-01-07 10:06:24. – http://www.helpmysql.org/en/thanks

[TAWAE08] Tyler Mitchell; Arnulf Christl; W. Lang, Jørgen; Astrid Riehl-Emde: *Web-Mapping mit Open Source-GIS-Tools: [aktualisiert, mit Tools-CD & farbigem Innenteil]*. 1. Aufl. Beijing [u.a.] : O´Reilly, 2008. – XXII, 454 S.. 1 CD-ROM (12 cm) : Ill., graph. Darst., Kt. – ISBN 978–3–89721–723–2

[WIS30] Open Source Matters Inc. (Hrsg.): *WISroGIS mapping - Joomla! Extensions Directory*. 2011-01-07 09:57:30. – http://extensions.joomla.org/extensions/maps-a-weather/maps-a-locations/maps/5414

[YAP42] Yahoo! Inc. (Hrsg.): *Yahoo! Developer Network*. 2011-02-01 14:45:42. – http://developer.yahoo.com/

Anhang A

Digitales Medium

Die Arbeit begleitendes digitales Medium mit folgenden Inhalten:

- **Bachelor Thesis** – Dokument, Bilder und Bildquellen
- **Benutzerhandbuch** – Dokument, Bilder und Quellen
- **Installation** – Joomla!-Installationspakete mit Quelldateien und benötigten JavaScript-Bibliotheken
- **SoSe 10** – Dokumentation zum Stand des Projektes im Sommersemester 2010
- **WiSe 10/11** – Protokolle und Datenbankentwürfe aus den Projektbesprechungen mit Bezug auf diese Arbeit.

(Medium befindet sich auf der nachfolgenden Seite)